ISSN 0265-2919
80p 28

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO

An ORBIS Publication
IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

[illegible]

CONTENTS

Next Week

- Inspired by the success of the Lisa and the Macintosh, Apple launch the IIc — a portable and somewhat upgraded version of the Apple II.
- Having completed the Interface Box, our workshop series moves on to building the external power supply and controller unit that accompanies it.
- In BASIC Programming we design a user-defined character generator utility for the Commodore 64, Spectrum and BBC Micro

QUIZ

1) How is the UNIX operating system manual unusual?
2) What is the disadvantage of MIDI's serial input/output transmission mode?
3) What is unusual about the Motorola 6809's accumulators?
4) Why was The Hobbit given away with The Hobbit?

Answers To Last Week's Quiz
1) MIT LOGO was the first version of the language to appear on microcomputers.
2) Although many of the 6809 registers are 16-bit, its 16-bit arithmetic and logic operations are severely limited — like the Z80, it is really just a powerful eight-bit processor.
3) Facsimile transmission, or 'fax', is the electronic transmission of images.
4) SQL is a query language used to access mainframe databases.

Editor Jim Lennox; **Managing Editor** Mike Wesley; **Art Director** David Whelan; **Technical Editor** Brian Morris; **Production Editor** Catherine Cardwell; **Art Editor** Claudia Zeff; **Chief Sub Editor** Robert Pickering; **Designer** Julian Dorr; **Art Assistant** Liz Dixon; **Editorial Assistant** Stephen Malone; **Sub Editor** Steve Mann; **Researcher** Melanie Davis; **Contributors** Geoff Bains, Harvey Mellor, Mike Curtis, Steve Colwill, Steve Malone, Rory Forsyth, Jim Lennox, Geoff Nairn, Graham Storrs; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Chris Cooper; **Production Controller** Peter Taylor-Medhurst; **Circulation Director** David Breed; **Marketing Director** Michael Joyce; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; **© APSIF Copenhagen 1984; © Orbis Publishing Ltd 1984: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Artisan Press Ltd, Leicester**

HOME COMPUTER ADVANCED COURSE – Price UK 80p IR £1.00 AUS $1.95 NZ $2.25 SA R1.95 SINGAPORE $4.50 USA and CANADA $1.95
How to obtain your copies of HOME COMPUTER ADVANCED COURSE – Copies are obtainable by placing a regular order at your newsagent, or by taking out a subscription. Subscription rates: for six months (26 issues) £23.80; for one year (52 issues) £47.60. Send your order and remittance to Punch Subscription Services, Watling Street, Bletchley, Milton Keynes, Bucks MK2 2BW, being sure to state the number of the first issue required.
Back Numbers UK and Eire – Back numbers are obtainable from your newsagent or from HOME COMPUTER ADVANCED COURSE. Back numbers, Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT at cover price. AUSTRALIA: Back numbers are obtainable from HOME COMPUTER ADVANCED COURSE. Back numbers, Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G Melbourne, Vic 3001. SOUTH AFRICA, NEW ZEALAND, EUROPE & MALTA: Back numbers are available at cover price from your newsagent. In case of difficulty write to the address in your country given for binders. South African readers should add sales tax.
How to obtain binders for HOME COMPUTER ADVANCED COURSE – UK and Eire: Please send £3.95 per binder if you do not wish to take advantage of our special offer detailed in Issues 5, 6 and 7. EUROPE: Write with remittance of £5.00 per binder (incl. p&p) payable to Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT. MALTA: Binders are obtainable through your local newsagent price £3.95. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Miller (Malta) Ltd, M.A. Vassalli Street, Valletta, Malta. AUSTRALIA: For details of how to obtain your binders see inserts in early issues or write to HOME COMPUTER ADVANCED COURSE BINDERS, First Post Pty Ltd, 23 Chandos Street, St. Leonards, NSW 2065. The binders supplied are those illustrated in the magazine. NEW ZEALAND: Binders are available through your local newsagent or from HOME COMPUTER ADVANCED COURSE BINDERS, Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington. SOUTH AFRICA: Binders are available through any branch of Central Newsagency. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Intermag, PO Box 57394, Springfield 2137.
Note – Binders and back numbers are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK market only and may not necessarily be identical to binders produced for sale outside the UK. Binders and issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.

COVER PHOTOGRAPHY BY IAN McKINNELL

JAPANESE TAKEAWAY

Pocket computers are used for a variety of serious applications. Engineers use them on-site to calculate stress and loading factors. On the floor of the Stock Exchange they are used by speculators to calculate profits, and they provide salesmen with accurate estimates of cost. We compare several machines from Casio's range.

Programmable calculators can be used in many similar applications, but pocket computers have a definite advantage in that they use BASIC and can handle text, whereas programmable calculators use their own special languages (often resembling machine code) and are suitable for numeric work only. Pocket computers are ideal for tasks in which calculations involving a set formula are regularly needed. Such calculations are often repetitive and tedious; pocket computer users can write their own programs to handle these.

The cheapest of Casio's well-established range of pocket computers is the FX-720P, which costs £60. Its dimensions are 165 x 85 x 15 mm ($6\frac{1}{2}$ x $3\frac{1}{4}$ x $\frac{1}{2}$ in) and it weighs only 210 grams ($7\frac{1}{2}$ oz). The FX-720P has a keyboard a little over half the size of a normal keyboard. It uses the conventional QWERTY layout, with the exception that the keys line up with those in the other rows, rather than being offset in the usual way.

Each alphabetic key can produce two other characters (such as punctuation marks) and BASIC keywords, when used with the Shift or Function keys. Unlike other keyboards, however, these keys are not used simultaneously with the alphabetic key; they are pressed and released before the appropriate key is pressed. This makes it easy to work the keyboard using one hand (leaving the other hand free to hold the computer).

The FX-720P uses a liquid crystal display (LCD) with 12 characters in a single line, and longer lines are displayed by using two cursor keys to scroll left or right. A small thumbwheel at the side of the display adjusts the LCD contrast to suit different viewing angles and levels of illumination. If the computer is not used for six minutes, it shuts the display off to save power.

The FX-720P is sold with a mere two Kbytes of memory. However, all user memory is supplied in the form of plug-in cartridges, which retain their contents when removed from the machine. A user can therefore store different programs and data on separate cartridges, which are plugged in when needed. Extra two Kbyte cartridges cost £25 each, and four Kbyte modules are sold at £36 each.

As the cartridges can hold so little data they quickly fill up, especially as the computer allows up to 10 BASIC programs to be held in memory at once. A special mode must be selected to enter a BASIC program and another is needed to run it. The FX-720P's version of BASIC is surprisingly good for such a small machine: almost all the standard commands and functions are included, with the exception of CHR$ and ASC. A simple command, BEEP, allows two different sounds to be produced.

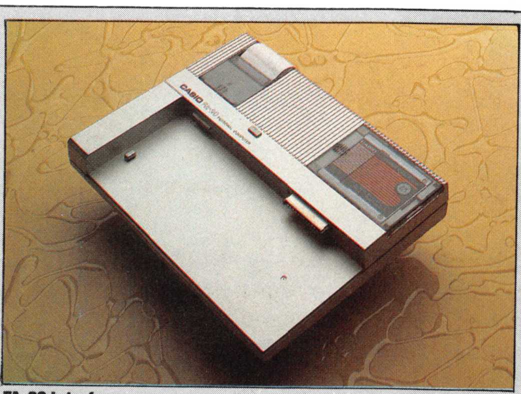
The FX-720P is regarded as being at the bottom of the Casio pocket computer range, and so most of its functions are improved upon by other models. One feature, however, is unique to this machine. This is the 'Data Bank', which is a small database program. Information such as names and addresses, expenses or diary dates can easily be entered using this program. The program will search for any item, and even allows for different levels of search — for example, if a search has found a large number of items it can search within them on another criteria. This makes it a direct rival to the Psion Organiser (priced at £100), yet the FX-720P costs less and is more versatile.

CASIO FX-750P AND PB-700

While the FX-720P seems to be biased towards the businessman, the next model in Casio's range, the FX-750P, is geared towards the needs of scientists and engineers. At a cost of £100, it offers a 24-character display and a slightly improved keyboard. It comes with a two Kbyte memory

Budget Model
The FX-720P pocket computer from Casio has 2K of memory on plug-in cartridges, a one-line liquid crystal display, and a full set of alphabetical and numeric keys. Additional cartridges can be plugged in to provide extra memory. The price of the FX-720P is £60

CHRIS STEVENS



CHRIS STEVENS

FA-20 Interface
Each of Casio's pocket computers can be attached to an interface of this type. This model allows the FX-750P to slide neatly into the recessed space. The FA-20 comes with rechargeable batteries and a mains adaptor for £80

cartridge, of the same type used by the FX-720P, but it has a slot for a second cartridge as well. This means it can have its memory capacity expanded to up to eight Kbytes. The machine is 185 mm (7¼ in) wide and weighs a mere 250 grams (9 oz). Unlike the FX-720P, the Shift and Function keys have to be used at the same time as the alphabetic key.

Ten of the keys have important constants programmed into them: the velocity of light, gravitational acceleration of the Earth, Planck's constant, Boltzmann's constant, Avogadro's constant, the elementary charge, the atomic mass, the electron mass, the gravitational constant and the molar volume. Physicists and chemists use these numbers frequently in calculations and having them fixed in memory saves the trouble of having to remember them and key them in. The computer also includes functions not normally found in BASIC: six hyperbolic functions and a set of statistical functions including standard deviation, linear regression and correlation. The BASIC is improved in several other ways so that it would not be out of place on many home computers.

The third pocket computer from Casio is the recently-launched £139 PB-700. This has a 20 character by four line display and four Kbytes of memory. The memory can be expanded up to 16 Kbytes using memory packs that plug in the underside of the computer. It is nearly 200 mm (8 in) wide and weighs 315 grams (11 oz). Like the other pocket computers it comes with a soft case to protect it.

The PB-700 has a similar keyboard to the FX-750P, except that it has a Caps key instead of a Function key. While this key is held down the computer produces lower case letters. The machine's version of BASIC is similar to that of the FX-750P, although it does have a few extra commands to draw graphics on the LCD, which has a resolution of 32 by 160 dots.

The BASIC also includes commands to draw graphics using the optional colour printer/plotter and cassette interface unit called the FA-10. This clips onto the computer and costs £194. It uses four different coloured ball pens to draw text and graphics on paper 115 mm (4½ in) wide. The BASIC commands allow the printer/plotter to draw lines, circles and axes for graphs.

The FA-10 also has a cassette interface built into it, allowing it to use an ordinary cassette recorder to save programs and data. Considering how small the computer's memory is, this is a very useful facility. For an extra £70, a tiny cassette recorder can be fitted into the printer/plotter giving a complete computer system small enough to be carried anywhere. A separate Centronics interface is also available (priced at £50) to allow standard computer printers to be used with the PB-700.

Cassette interfaces and printers are also available for the FX-720P and FX-750P. The FX-720P uses the FP-12S printer. A separate unit, the FA-3 acts as an interface for a cassette recorder. The FX-750P uses the £80 FA-20, which is a tiny printer and a cassette interface combined. It has a space to store a memory card and comes with rechargeable batteries, a mains transformer and a protective case.

All three computers come with extensive manuals. These contain all that is needed but they are obviously translated from Japanese and are oddly worded in many places. They are not unusable, but they can be confusing to the beginner. Another problem is that there is almost no software available for these computers. Owners will have to write their own or copy the sample programs listed in each machine's manual.

PRODUCT	PRICE	SIZE	WEIGHT	MEMORY	DISPLAY	REMARKS
FX-720P	£60	165x85x15mm	210gm	2K	1 x 12 ch	Built-in database. Numeric keypad
FX-750P	£100	185x85x15mm	250gm	2K	1 x 24ch	Second cartridge slot. Physical constants programmed in. Expanded BASIC
PB-700	£139	200x85x15mm	315gm	4K	4 x 20 ch	Graphics capability. Printer/plotter available

PLAYING TURTLE

Our LOGO series has so far concentrated on using the language in its 'immediate mode', in which each command is obeyed as it is entered. An alternative mode allows LOGO users to define a sequence of commands as a 'procedure' that is given a specific name. The commands in the procedure will be carried out only when it is called by name.

Procedures may be unfamiliar to many micro programmers, but the concept is found in everyday 'instruction procedures' such as cooking recipes and knitting patterns. Here we'll create a LOGO procedure that we will call SQUARE. (Unlike LOGO commands, which must always be typed in upper case letters, procedure names may be in either upper or lower case.) We begin by typing:

```
EDIT SQUARE
```

The screen will clear, then the message TO SQUARE will appear at the top of the display as a reminder of the procedure name, and EDIT CTRL-C TO DEFINE CTRL-G TO ABORT will be printed at the bottom. This somewhat cryptic message helps to guide your actions while in 'edit' mode. EDIT simply tells you that LOGO is no longer in immediate mode but expects a procedure to be created. Once this is done and you are ready to store your procedure, pressing Control-C will define (record) the listing, while Control-G will allow you to abort the procedure and start again.

You may type whatever you wish in edit mode, but none of the commands will be obeyed at this time. LOGO simply makes a note of your instructions and stores them in its 'dictionary' under the name you have given your procedure. Complete the SQUARE procedure definition by typing the following commands:

```
TO SQUARE
  REPEAT 4 [FD 50 RT 90]
END
```

Now you can tell LOGO to define this procedure (i.e. remember it) by typing Control-C; you will then receive the message SQUARE DEFINED. If you have made a mistake in your definition you can abort the whole process by typing Control-G, in which case you must begin again, or you may use the edit mode's full screen editor to make corrections. This allows you to use the cursor keys to move anywhere within the procedure and insert or delete characters where desired. (The immediate mode's line editor permits corrections to be made only on the last line entered). LOGO has many more commands that make editing long procedures simple; we will consider these later.

Now your procedure has been successfully defined, let's see how it works. Type DRAW to access the graphics screen, then type SQUARE — the commands in the procedure definition will now be obeyed and the turtle will draw the square. As you can see, procedures are used in exactly the same way as the basic LOGO commands — simply type the procedure name and it will carry out the instructions you have defined. The original commands that are present when LOGO is loaded are called 'primitives'. Once you have 'taught' the language a new procedure it can then be used in just the same way as a primitive. In other words, LOGO is an 'extensible' language, so it may be tailored to suit your own particular requirements.

If your procedure does not do what you expected, it is a simple matter to change the definition. As we have defined it, SQUARE draws a square with sides of 50 units in length. Let's assume that you would prefer a smaller square — say, 30 units each side. Return to the editor by typing in:

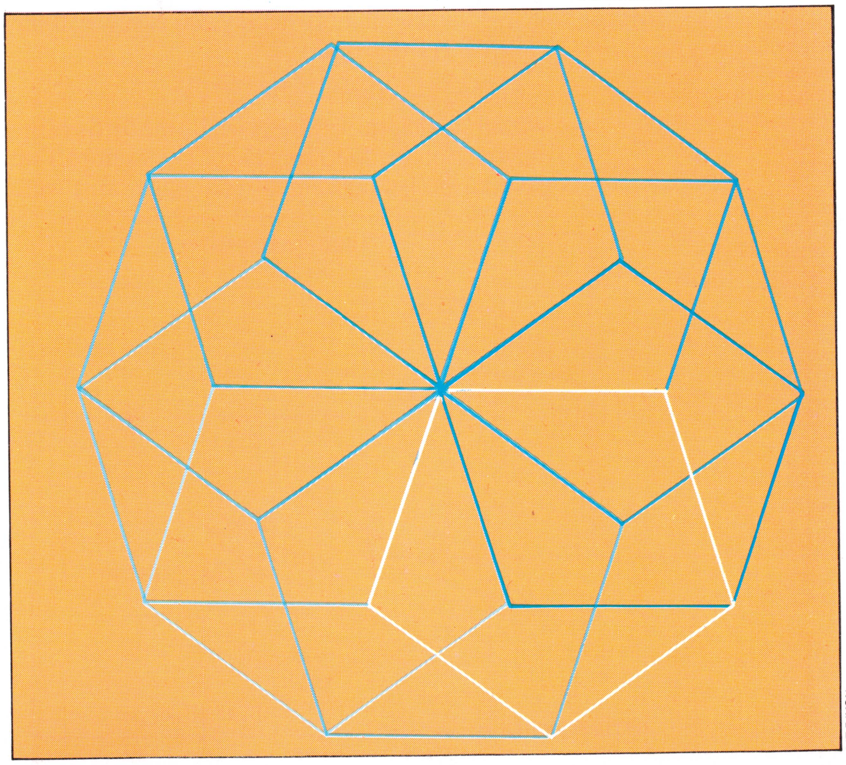
```
EDIT SQUARE
```

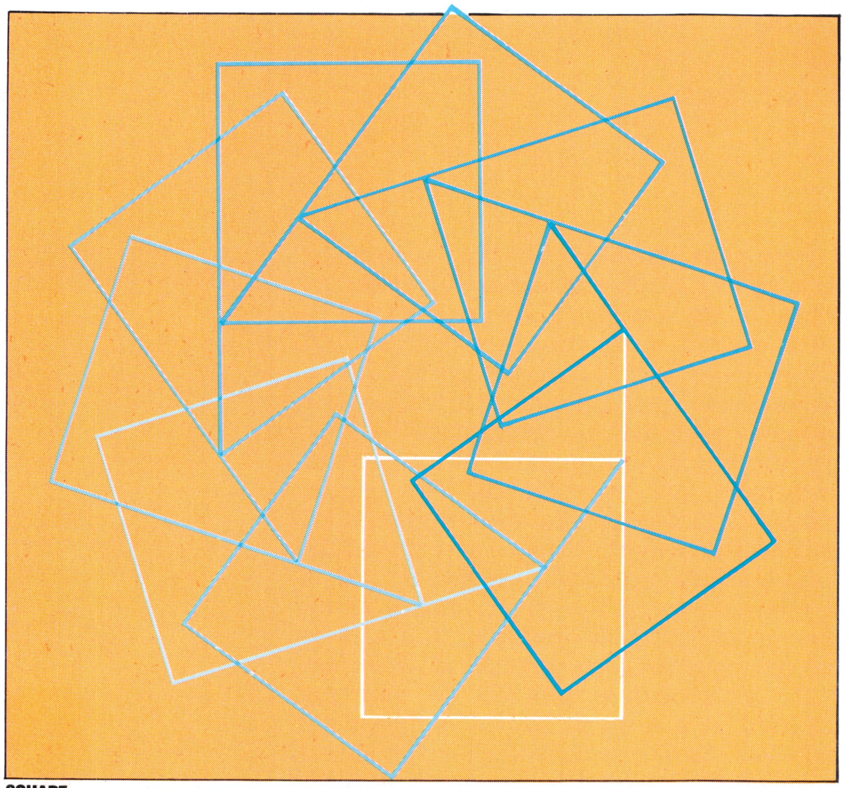
The text making up the procedure is now displayed, and the screen editor can be used to change the 50 to 30. Once this has been done, redefine the procedure with Control-C and type SQUARE to ensure that it is now the size you wanted. As an exercise, try writing procedures to

Abbreviations

EDIT	ED
HIDETURTLE	HT
SHOWTURTLE	ST

PENTAGON





SQUARE

draw the shapes you created in the last instalment (see page 532): triangles, rectangles, pentagons, stars, etc.

Now try using your procedures with the REPEAT command. For example, define PENT as:

```
REPEAT 5 [FD 50 RT 72]
```

and then try:

```
REPEAT 10 [PENT RT 36]
```

You can experiment with other shapes in the same way, and you should soon discover that complex shapes can be built up quickly and easily by using REPEAT. Try this one:

```
REPEAT 10 [SQUARE RT 36 FD 25]
```

We show what results are given by the turtle in the diagrams on the following page. The examples we gave in the last instalment (see page 532) were designed for you to experiment with; your attempts at drawing the various shapes may well have led to some unexpected results. To draw a triangle, you may first have tried something like this:

```
REPEAT 3 [FD 50 RT 60]
```

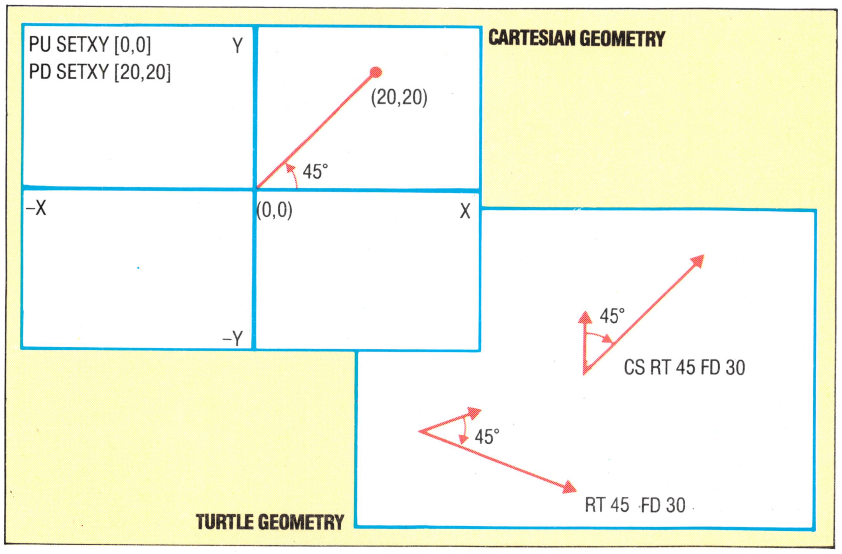
and then discovered that you have created half a hexagon! If you tried to draw the six-pointed star, you will certainly have found it much harder than you expected. When dealing with such problems, it is often helpful to 'play turtle' by imagining that *you* are the turtle moving around. Indeed, it is said that you can tell the difference between BASIC and LOGO programmers by watching their shoulders move while they program! Imagine drawing any closed shape from the turtle's point of view. The turtle has to go completely around the shape, and must end up back in the same place, facing in the original direction. So it must turn through a multiple of 360 degrees. If the shape is 'convex' (none of its internal angles is greater than 180 degrees), then the turtle will have turned through exactly 360 degrees. In the case of the triangle, there are three turns to be made — so each must be 360/3 = 120 degrees. From this, you should be able to deduce that the correct command for drawing a triangle is:

```
REPEAT 3 [FD 50 RT 120]
```

Even for non-convex polygons, such as the star shapes we have already investigated, the same principle applies — the only difference is that now the total angle turned is a whole-number multiple of 360 degrees (because you are creating more than one complete shape).

These principles are general (they don't apply simply to polygons) and they make up what is known as the 'total turtle trip theorem'. If you examine the six-pointed star from the turtle's point of view, you will see that the shape cannot be drawn by the turtle simply moving forward and turning through the same angle each time. Instead, a more complex procedure is required.

Cogito Ergo LOGO
The turtle can be addressed in terms of Cartesian or 'Turtle' geometry. In the former, turtle positions are expressed absolutely — they are measured from the imaginary X and Y axes whose origin [0,0] is the CS position; in turtle geometry, commands are expressed relative to the turtle's current position and heading, whatever they may be



TURTLE CO-ORDINATE GEOMETRY

Turtle geometry is concerned with the properties of shapes themselves, rather than the relationship of shapes to an external point of reference, as is the case with co-ordinate geometry. Turtle geometry is relative — movements are made in a specified number of units from the turtle's current position on the screen. Co-ordinate geometry uses absolute values — the screen is imagined as a grid, with a defined number of units extending vertically and horizontally from the centre. Each point on the grid has a specific numeric value, and movement is defined in relation to these grid references. However, it is possible to use co-ordinate geometry with the turtle.

For example, the command SETXY 20 30 moves the turtle from its current position to the point (20,30). If PENDOWN has been specified, the turtle will draw a line as it moves; conversely, the PENUP command will cause the turtle to move without leaving a trace. The origin (0,0) is in the centre of the screen.

The following procedures perform the same task, and illustrate the difference between turtle and co-ordinate geometry:

```
TO SQUARE1
   REPEAT 4 [FD 50 RT 90]
END
TO SQUARE2
   SETXY 0 50
   SETXY 50 50
   SETXY 50 0
   SETXY 0 0
END
```

Typing SQUARE1 or SQUARE2 after returning to DRAW will give exactly the same result. But what happens if you wish to rotate the two squares through 30 degrees? RT 30 SQUARE1 works correctly in the first case, but the second procedure needs to be completely rewritten (because it specifies absolute screen co-ordinates), and this is not a trivial task. But there are times when the use of co-ordinates in LOGO is helpful: as you will see from these examples, SETXY is considerably faster at drawing lines than FORWARD is.

Another feature of turtle geometry is its 'short-sightedness'. The turtle concerns itself only with a single movement at a time; it builds up shapes by taking a series of short 'steps'. Let's play turtle and consider how a circle is constructed. Imagine that you are the turtle — what would you need to do to produce a circle shape? You would move forward for a short distance and turn a little — and you would repeat this sequence many times. In LOGO terms, this translates into something like:

```
TO CIRCLE
REPEAT 360 [FD 1 RT 1]
END
```

This procedure works, but it runs very slowly. It can be made to run faster if the turtle is not drawn at every step. The LOGO command HIDETURTLE is used to make the turtle invisible — SHOWTURTLE draws it again. Our example is really drawing a 360-sided polygon: the lines that form it are so short that they give the appearance of a smooth curve. In fact, 360 sides are more than enough to give the illusion of a circle — a 36-sided polygon is sufficient. So the following procedure draws an acceptable circle, and draws it at a considerably greater speed:

```
TO CIRCLE
   REPEAT 36 [FD 10 RT 10]
END
```

Now try writing procedures that will result in a semi-circle, a quarter-circle, and a sixth of a circle; and combine two of these to make a petal shape.

Playing turtle again, how would you move if you wished to spiral in towards a point, rather than travelling in a circle around it? You would still move a short distance before turning, but in this case you would turn more and more each time (or travel a shorter distance for each turn, which comes to the same thing). The spiral procedure here is a little long-winded, as we have restricted it to use only the commands we have introduced to date, but it will demonstrate that principle:

```
TO SPIRAL
   FD 10 RT 10 FD 10 RT 20 FD 10 RT 30
   FD 10 RT 40 FD 10 RT 50 FD 10 RT 60
   FD 10 RT 70 FD 10 RT 80 FD 10 RT 90
END
```

ALAN COODE COURTESY OF VALIANT DESIGNS LTD.

LOGO Demo
Anthony Ginn's Demolition Turtle game introduces the floor turtle and helps to encourage spatial awareness in young children. One player places a toy tower somewhere on the board, and chooses the turtle's start position and heading. The other player then programs the turtle to hit the tower, using as few commands as possible

Logo Flavours

LOGO editors are very similar, but each has its own peculiarities because of the particular keyboard of the machine. Consult the LOGO manual for your machine.

For all LCSI versions the command to edit the procedure SQUARE is EDIT "SQUARE (quotation mark before but not after the name of the procedure).

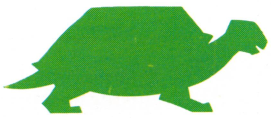
To set the turtle at (20,30) use SETPOS [2 30] on all LCSI versions.

Exercise Answers

Triangle	REPEAT 3 [FD 50 RT 120]
Pentagon	REPEAT 5 [FD 50 RT 72]
Hexagon	REPEAT 6 [FD 60 RT 60]
Rectangle	REPEAT 2 [FD 25 RT 90 FD 50 RT 90]
Parallelogram	REPEAT 2 [FD 25 RT 70 FD 50 RT 110]
Rhombus	REPEAT 2 [FD 50 RT 70 FD 50 RT 110]

Stars:

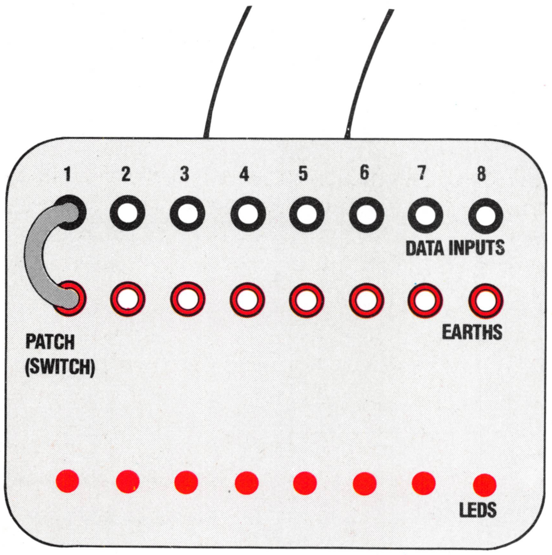
```
1. REPEAT 5 [FD 50 RT 144]
2. REPEAT 8 [FD 50 RT 135]
3. RT 30 REPEAT 3 [FD 50 RT 120]
   PU LT 30 FD 29 RT 90 PD
   REPEAT 3 [FD 50 RT 120]
4. REPEAT 10 [FD 50 RT 108]
```



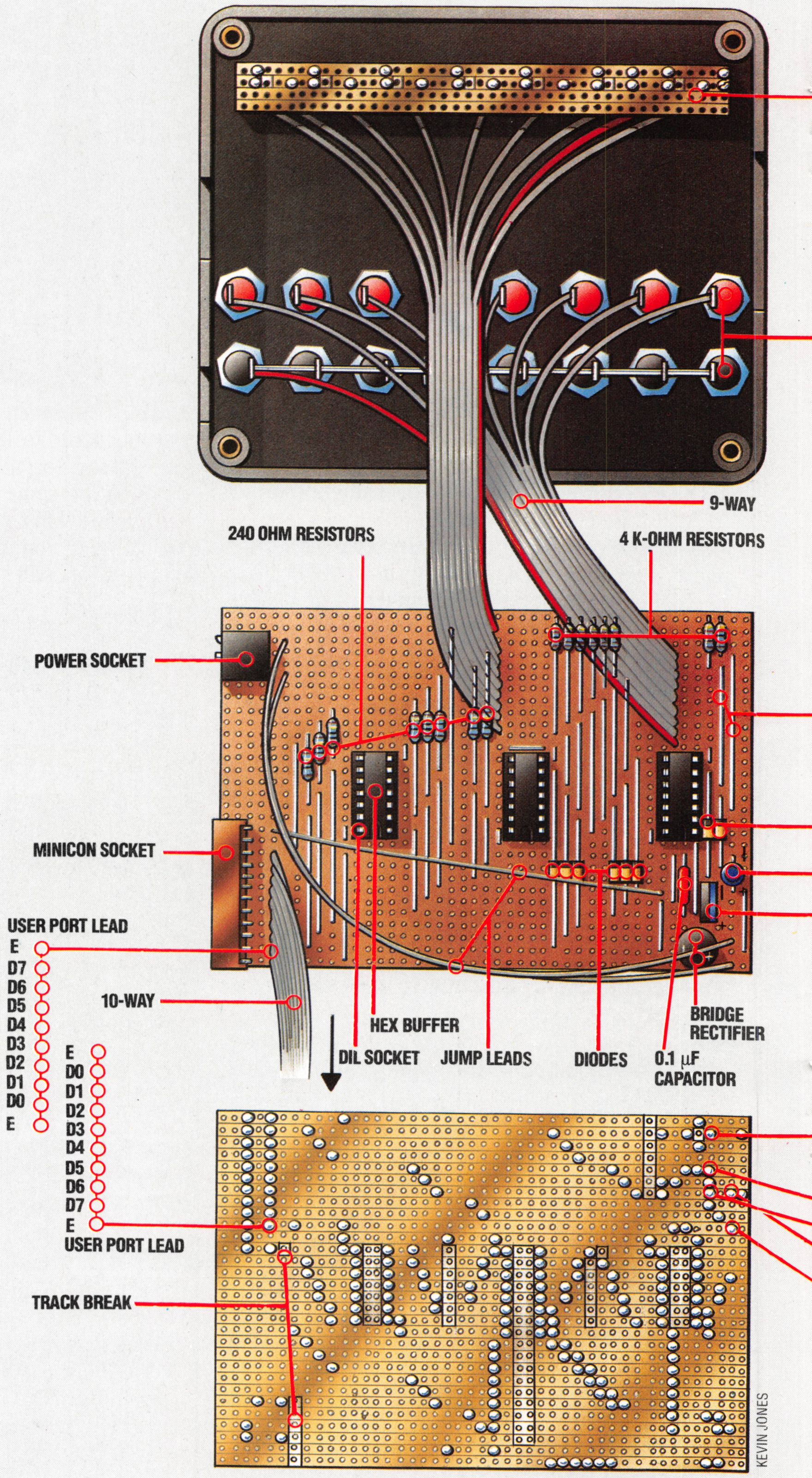
ALL BOXED UP

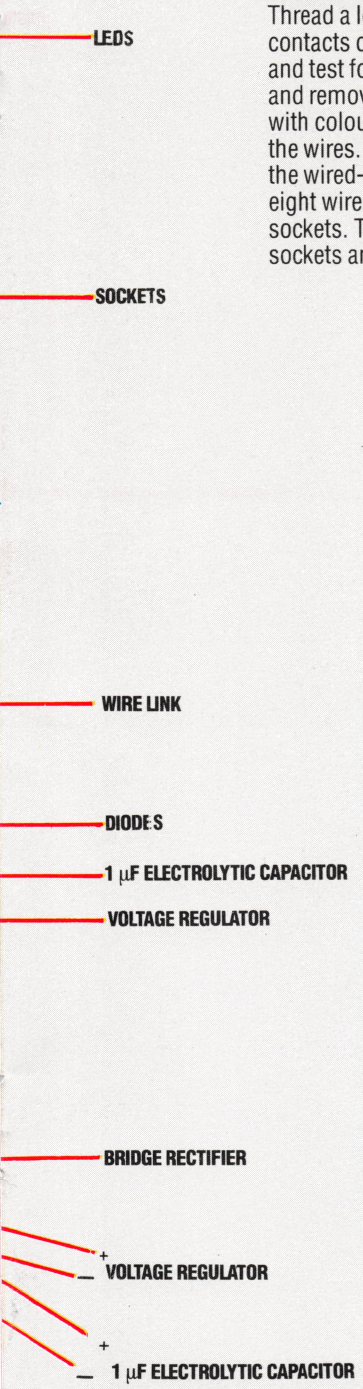
In our Workshop series we have discussed the functions of the Commodore 64 and BBC Micro user ports, and have shown you how to construct a buffer box to handle input and output. Now it is time to test the box in operation…

Once the input buffer box with LEDs has been constructed, we can write some simple software to test its operation by using it as a reaction timer. Here, we shall use bit 7 of the data register for the input switch and bits 0 to 6 for the LEDs. The objective is to time the interval between the computer turning the LEDs on and a switch being pressed. We must first connect a switch between input terminal 7 on the box and the corresponding earth; this switch can be of any type as it is to be used only to make and break the circuit between bit 7 of the data register and earth. A patch cord, for example, can be used as a switch. Connect the switch, ensuring that it is breaking the circuit between bit 7 and earth (the value of bit 7 should be one). The following diagram shows the connections:



Our test program first sets up the data direction register so that bit 7 is set for input and all others are set for output, and then zeros the data register. After a random delay, seven of the LEDs are lit and the computer's internal timer is initialised. Timing continues until bit 7 is sent low when the connection is made. Note that the LEDs are wired in such a way as to light when a zero is present in the data register and go out when a one is present.





Inside The Box

Thread a length of tinned wire as shown through the contacts of the sockets. Solder it to each contact, and test for continuity. Take 20 cm of ribbon cable and remove three wires, leaving a nine-way ribbon with coloured edge stripe. Bare and tin the ends of the wires. Solder the coloured wire to the leftmost of the wired-up sockets. Now solder the remaining eight wires in order to the contacts of the other sockets. Test for continuity between each of the sockets and the end of the wire connected to it

Building The Circuit Board

The board shown has 30 tracks with 45 holes and fits our box exactly. Follow the layout illustrations carefully, and you should have no trouble building the board. Use as little solder as possible, and take care not to bridge the tracks; check continually that you are placing the right components in the right places. The diodes, electrolytic capacitor, bridge rectifier and voltage regulator must all be connected in the direction shown — any other orientation will damage them, so study their plus/minus markings. All components are heat-sensitive, so don't 'over-cook' them with the iron. When fitting the minicon and power sockets take care to locate the pins in the right board holes, but don't bend the pins roughly. Use the wires you removed from the ribbon cable for the 'jump leads'.

When everything is in place on the board, cut the copper tracks exactly as shown. You can buy a special tool for this, or you can hold a drill bit in your fingers and twirl it in a hole, cutting the copper gradually. Don't leave shreds of copper on the board. Solder the ribbon cables onto the board. The orientation of the socket and LED leads is shown by the coloured stripe, but the user port lead needs some thought — the two earth lines must be in holes 1 and 10 (counting from the edge of the board), and the signal lines must go in order into holes 2 to 9 so that the least significant line is closest to the edge of the board.

Lastly, using the board as a template, cut slots in the box sides to accommodate the sockets and the user port lead

```
 10 REM * BBC REACTION TIME **
 15 :
 20 DDR=&FE62: DATREG=&FE60
 30 ?DDR=127: REM LINES 0-6 OUTPUT
 40 ?DATREG=127: REM LEDS OFF
 50 :
 60 CLS: PRINT "GET READY"
 70 DELAY=3000+RND(9000)
 80 FOR I=1 TO DELAY:NEXT:REM DELAY LOOP
 85 FOR D=1 TO 200:NEXT D
 90:
 97 REPEAT UNTIL ?DATREG AND 128=1
100 ?FE60=0: REM TURN LEDS ON
110 TIME =0: REM INIT TIMER
120 REPEAT
130 UNTIL ?DATREG AND 128=0: REM S/W ON
140 :
150 PRINT "TIME TAKEN="TIME/100"SECS"
160 END
```

The program makes use of TIME, a reserved variable that returns a value corresponding to the number of hundredths of a second since TIME was last set to zero. Making use of the logical AND in line 130 isolates bit 7 (value 128) so that it can be tested independently of the other bits in the data register (see page 66). When the switch is thrown, the value of a bit 7 changes from one to zero.

A similar program can be written for the Commodore 64 using its internal timer, TI. TI works differently from TIME, returning a value in sixtieths of a second since the machine was turned on. To use it we must take the value of TI at the start of the interval to be timed and subtract it from the value of TI at the end.

```
 10 REM CBM 64 ** REACTION TIMER **
 20 :
 30 DDR=56579: DATREG=56577
 40 POKE DDR,127: REM LINES 0-6 OUTPUT
 50 POKE DATREG,127: REM LEDS OFF
 60 :
 70 PRINT CHR$(147): REM CLEAR SCREEN
 80 PRINT "GET READY"
 90 DE=3000+INT(9000*RND(1))
100 FOR N=1 TO DE:NEXT: REM DELAY LOOP
110 :
120 POKE DATREG,0: REM TURN LEDS ON
130 T=TI: REM TAKE START TIME
140 IF PEEK(DATREG) AND 128<>0 THEN 140
150:
160 TM=(TI-T)/60: REM CALC INTERVAL
170 PRINT "TIME TAKEN=";TM;"SECS"
180 END
```

In future instalments of the course, we will look at the construction of higher current outputs that are sufficient to drive electric motors, and we will design software to control bi-directional and variable speed motors.

Experiments For You To Try

1) Write a program that will light one LED at a time in sequence from left to right.
2) Write a program to make the LEDs light in sequence on lines 0 to 6 (as in question 1), but include a switch on line 7 to change the sequence direction. Can you alter your program to operate with a 'train' of three LEDs?
3) Write a program to simulate a dice throw, using six LEDs and one switch.
4) Write a program to simulate the action of a traffic light, using three LEDs.
5) Write a program to count the number of 'cars' (pulses on a switch) arriving while a traffic light is on red and to change the lights when the number of cars exceeds 10 or if one minute has elapsed since the last change.

Buffer Box Answers

These are possible solutions to the Exercises on page 547:

LED Sequencing (I)

```
 10 REM CBM 64 VERSION 2.1
 20 DDR=56579: DATREG=56577
 30 POKE DDR,255: REM ALL OUTPUT
 40 POKE DATREG,255: REM ALL LEDS OFF
 50 GET A$
 60 FOR N=1 TO 7
 70 POKE DATREG,255-(2^N)
 80 NEXT N
 90 IF A$="" THEN 50
100 END
```

```
 10 REM BBC VERSION 2.1
 20 DDR=&FE62:DATREG=&FE60
 30 ?DDR=255: REM ALL OUTPUT
 40 ?DATREG=255: REM ALL LEDS OFF
 50 REPEAT
 60 A$=INKEY$(1)
 70 FOR N=0 TO 7
 80 ?DATREG=255-(2^N)
 85 FOR D=1 TO 200: NEXT D
 90 NEXT N
100 UNTIL A$<>""
110 END
```

LED Sequencing (II)

```
 10 REM CBM 64 VERSION 2.2
 20 DDR=56579: DATREG=56577
 30 POKE DDR,127: REM L7 INPUT
 40 POKE DATREG,255: REM ALL LEDS OFF
 50 GET A$
 60 FOR N=0 TO 7 STEP S
 70 POKE DATREG,255-(2^N)
 80 NEXT N
 90 IF PEEK (DATREG) AND 128=0 THEN S=-1
100 IF PEEK(DATREG) AND 128=1 THEN S=1
110 IF A$="" THEN 50
120 END
```

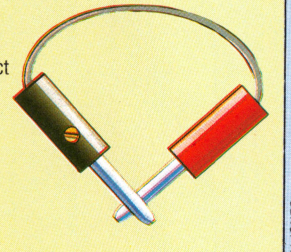
```
 10 REM BBC VERSION 2.2
 20 DDR=&FE62:DATREG=&FE60:S=1
 30 ?DDR=127: REM L7 INPUT
 40 ?DATREG=255: REM ALL LEDS OFF
 50 REPEAT
 60 A$=INKEY$(1)
 70 FOR N=0 TO 7 STEP S
 80 ?DATREG=255-(2^N)
 85 FOR D=1 TO 200:NEXT D
 90 IF DATREG AND 128=0 THEN S=-1 ELSE S=1
100 NEXT N
110 UNTIL A$<>""
120 END
```

LED Sequencing (III)

For the train of three LEDs, make the following changes to answer (2):

```
CBM VERSION
60 FOR N=2 TO 7 STEP S
70 POKE DATREG,255-((2^N)+2^(N-1)+2^(N-2))

BBC VERSION
70 FOR N=2 TO 7 STEP S
80 ?DATREG=255-((2^N)+2^(N-1)+2^(N-2))
```

PATCH CORD
Use 8 cm of insulated wire (a single line from the ribbon cable, for example) to connect a red plug to a black plug. This is called a patch cord, and is used to 'patch' one socket to another. Make eight such cords



KEVIN JONES

Dice Throwing

```
 10 REM CBM 64 VERSION 2.3
 20 DDR=56579: DATREG=56577
 30 POKE DDR,127: REM L7 INPUT
 40 POKE DATREG,255: REM ALL LEDS OFF
 50 GET A$
 60 IF PEEK(DATREG) AND 128<>0 THEN 60
 70 NR=INT(RND(1)*6)+1: REM SELECT NO.
 80 POKE DATREG,255-((2^NR)-1)
 90 IF PEEK(DATREG) AND 128<>1 THEN 90
100 IF A$="" THEN 50
110 END
```

```
 10 REM BBC VERSION 2.3
 20 DDR=&FE62:DATREG=&FE60:S=1
 30 ?DDR=127: REM L7 INPUT
 40 ?DATREG=255: REM ALL LEDS OFF
 50 REPEAT
 60 A$=INKEY$(1)
 70 REPEAT UNTIL (?DATREG AND 128)=0
 80 NUMBER=RND(6): REM SELECT NUMBER
 85 FOR D=1 TO 200:NEXT D
 90 ?DATREG=255-((2^NUMBER)-1)
100 REPEAT UNTIL ?DATREG AND 128=1
110 UNTIL A$<>""
120 END
```

Traffic Lights (I)

```
 10 REM CBM 64 VERSION 2.4
 20 DDR=56579: DATREG=56577
 30 POKE DDR,255: REM ALL OUTPUT
 40 POKE DATREG,255: REM ALL LEDS OFF
 45 REM BIT2=RED,BIT1=AMBER,BIT0=GREEN
 50 RD=255-4: AM=255-2:GN=255-1
 60 GET A$
 70 POKE DATREG,RD
 80 FOR N=1 TO 200:NEXT:REM DELAY LOOP
 90 POKE DATREG,RD+AM
100 FOR N=1 TO 40:NEXT:REM DELAY LOOP
110 POKE DATREG,GN
120 FOR N=1 TO 200:NEXT: REM DELAY LOOP
130 IF A$="" THEN 60
140 END
```

```
 10 REM BBC VERSION 2.4
 20 DDR=&FE62:DATREG=&FE60:S=1
 30 ?DDR=255:REM ALL OUTPUT
 40 ?DATREG=255: REM ALL LEDS OFF
 50 REM BIT2=RED, BIT1=AMBER,BIT0=GREEN
 60 RED=255-4:AMBER=255-2:GREEN=255-1
 70 REPEAT
 80 A$=INKEY$(100)
 85 FOR D=1 TO 200:NEXT D
 90 ?DATREG=RED
100 FOR N=1 TO 200:NEXT:REM DELAY LOOP
110 ?DATREG=RED+AMBER
120 FOR N=1 TO 40:NEXT:REM DELAY LOOP
130 ?DATREG=GREEN
140 FOR N=1 TO 200:NEXT:REM DELAY LOOP
150 UNTIL A$<>""
160 END
```

Traffic Lights (II)

Make the following changes to answer (4):

```
CBM VERSION
30 POKE DDR,127: REM L7 INPUT
65 T=TI: REM INIT TIMER
75 IF PEEK(DATREG) AND 128=0 THEN C=C+1
76 IF PEEK(DATREG) AND 128<>1 THEN 76
77 IF C<=10 AND T-TI<3600 THEN 75


BBC VERSION
30 ?DDR=127: REM L7 INPUT
75 T=0:C=0
95 REPEAT
96 IF ?DATREG AND 128=0 THEN C=C+1
97 REPEAT UNTIL ?DATREG AND 128=1
98 UNTIL T>6000 OR C>10
```

E

EXPONENT

Mathematical equations can often become extremely complicated, so long arithmetical expressions are written in a simplified type of shorthand whenever possible. One convention for shortening lengthy, repetitive multiplications is to use an *exponent*. For example, look at the expression:

$$7\times7\times7\times7\times7 = 16{,}807$$

What the expression indicates is 'seven multiplied by seven a total of five times'. By using an exponent, written as a small numeral above and to the right of the number being multiplied, we can simplify this expression thus:

$$7\times7\times7\times7\times7 = 7^5 = 16{,}807$$

The numeral 5 is called the exponent or index, and 7 the base or mantissa. The shortened expression means the same as the longer version, but occupies much less space and can make arithmetic simpler. In multiplication, for example, the product of numbers with like bases is formed from the sum of the exponents: $49\times343 = 7^2\times7^3 = 7^5 = 16{,}807$.

FACSIMILE TRANSMISSION

The transmission of documents or images from one location to another is essential to the operation of the electronic office. Machines to carry out this task have been available for many years, but early *facsimile transmission* (also known as 'fax') was dependent on analogue electronics. Using this system, the piece of paper containing the image to be transmitted was taped to a rotating drum and scanned by a photocell — a process taking about five minutes to cover an A4 sheet. Electronic signals representing the image were then transmitted via the telephone network to the receiving station, where a similar rotating drum was used to convert the signals back into an image by using heat-sensitive printing. Newer systems use lasers to scan the document to be sent and employ digital techniques to transmit the data to the receiving station, where a laser printer produces a high-quality copy.

FACTORIAL

The *factorial* of a positive integer number is the product of all the integers from one to the specified number. The shorthand symbol for factorial is an exclamation mark following the number. For example:

$1! = 1$
$2! = 1\times2 = 2$
$3! = 1\times2\times3 = 6$
$4! = 1\times2\times3\times4 = 24$

It is seen most often with the combination and permutation operators, thus:

$${}^{n}C_{r} = \frac{n!}{(n-r)!\,r!} \text{ and } {}^{n}P_{r} = \frac{n!}{(n-r)!}$$

gives the number of ways of combining and permutating r things from n. For example, the number of possible five-card hands in a pack of 52 is ${}^{52}C_5$, and the number of ways of laying those hands out in order is ${}^{52}P_5$.

FAIL-SAFE

Computers designed for sensitive or dangerous applications — for example, the control of fuel rods in a nuclear power station — are designed to be *fail-safe*. This simply means that a fault in the system will not cause a fatal error. In practice, it is not possible to guard against all possible faults, but the use of back-up systems and safety procedures can ensure that any malfunction has a minimal effect on the overall system.

An example of a fail-safe system is the 'dead man's handle' used on some trains. This requires pressure to be applied to a lever at all times: if this pressure is relaxed, power is shut off and the brakes are applied automatically.

FAN-IN

A *fan-in* is the number of input lines used in a logic gate or a logic device.

FAN-OUT

A *fan-out* indicates the maximum number of output devices that can be safely driven by a logic gate. If the number of output devices attached to a logic gate exceeds the device's fan-out, the difference between the voltage levels that correspond to logical 1 and logical 0 is decreased. This makes it more likely for errors to occur in the operation of the logic gate.

FEEDBACK

Feedback is information taken into a control device from an external source, and is used by the system to determine the next action that needs to be taken. A simple example is a thermostat that controls the temperature in a house. The thermostat reads the temperature around it and activates the heating system when the temperature reaches a minimum point, or turns the system off when the system reaches a maximum point. Feedback can be used in control projects with microcomputers by having an analogue-to-digital convertor connected to an external sensor.

TOPPING THE CHARTS

The teenage programmer who makes thousands of pounds from writing games programs captures the imagination of many computer owners. In this article we follow up two of these success stories and see how a fortune can be made from an original programming idea.

In the summer of 1984 Gremlin Graphics launched a new computer game called Wanted: Monty Mole. The game is expected to be a bestseller. If so, then the programmer, Peter Harrap, stands to earn several thousand pounds. Yet Harrap is just 19 and Wanted: Monty Mole was his first commercial game.

Peter Harrap belongs to a small group of 'whizz kid' programmers, usually in their 'teens, who can earn considerable sums of money if they come up with a top-ten game. One 14-year-old made nearly £4,000 from writing games software for the BBC Micro. Another programmer, aged 17, was given a Lotus Esprit sports car by the software company he worked for.

Like most games programmers Harrap is self-taught. The first computer he owned was a Sinclair ZX81 and after seeing the small amount of software that was then available, he decided to have a go at writing his own games programs. To do this he had to learn Z80 machine code as BASIC is far too slow for arcade games. Harrap soon switched his attentions to the Sinclair Spectrum and concentrated on trying to improve one game that particularly interested him, Quicksilva's Ant Attack (see page 6). Harrap broke into the machine code and altered the 'landscape' featured in the program. His modified version was rejected by Quicksilva, but a local computer enthusiast, Ian Stewart, who was looking for new talent for his small software company was impressed by the tape Harrap sent him. Harrap's game infringed the copyright on the original Ant Attack so Stewart's company, Gremlin Graphics, did not accept it. However, the company needed a high-powered Spectrum programmer so they took on Harrap immediately.

Every software company looks for an interesting and unusual theme in a game that it hopes will push it up to number one in the software charts. Gremlin Graphics conceived the idea of Wanted: Monty Mole in the hope that its topical theme of the miners' strike would capture the public's imagination. The idea was put into practice by Harrap, who wrote a program for it to run on the Sinclair Spectrum.

The game took Harrap four months to program. He began by writing a sprite generator routine to enable the various graphics shapes to be designed quickly, he then drew the coal mine and finally he programmed in the rules of play that determine the characters' movements.

The game received nationwide television coverage because of its topicality, and was soon a bestselling hit. The media reports did not ignore the irony that Harrap's father, who worked at the local colliery, was himself on strike, nor the fact that Gremlin Graphics had pledged to donate five pence for every cassette sold to the Miners' Welfare Fund.

Gremlin Graphics pays Harrap a *royalty*, or a percentage of each cassette sold. These royalty deals vary from company to company. Most companies will pay royalties on a sliding scale: five per cent for the first 3,000 copies sold, for example, and then seven per cent for the next 10,000. The percentage will be calculated either on the nett price, which is the price the software company receives from the distributor, or on the gross price, which is the price paid by the public.

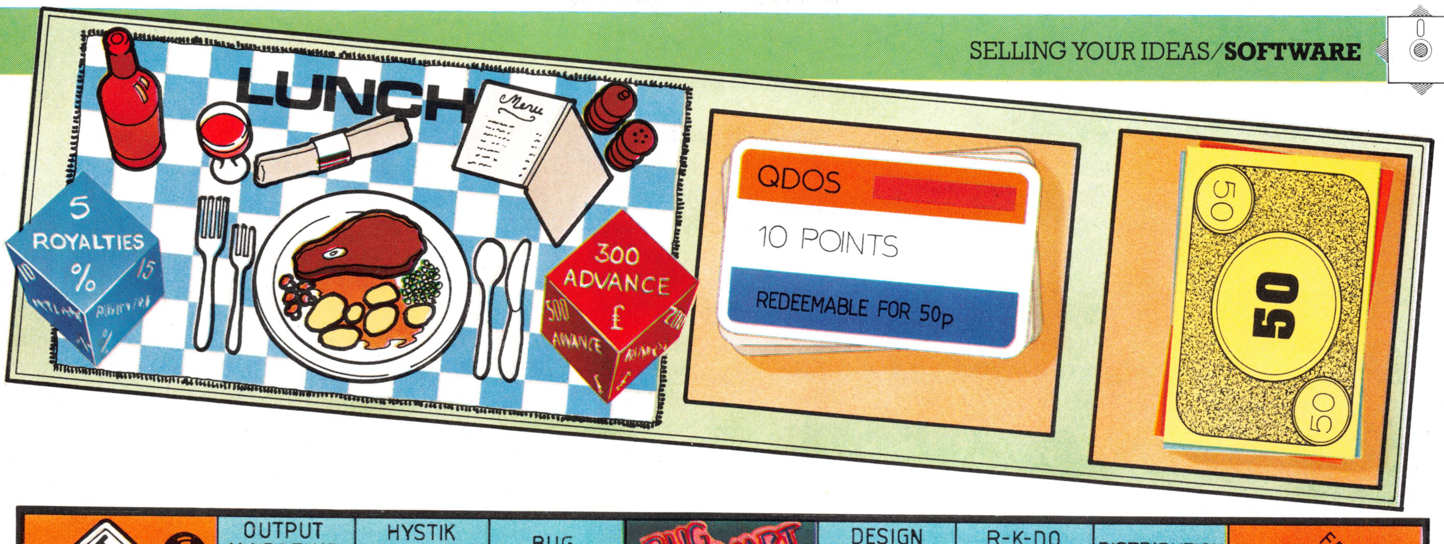
Most companies will also pay its programmers an *advance*. This is paid against royalties, in other words it will be subtracted from the first royalties paid. Many companies, however, will not pay their programmers a royalty and will opt instead for a single up-front payment. Programmers should be wary of such deals as they can result in lost revenue if sales are high.

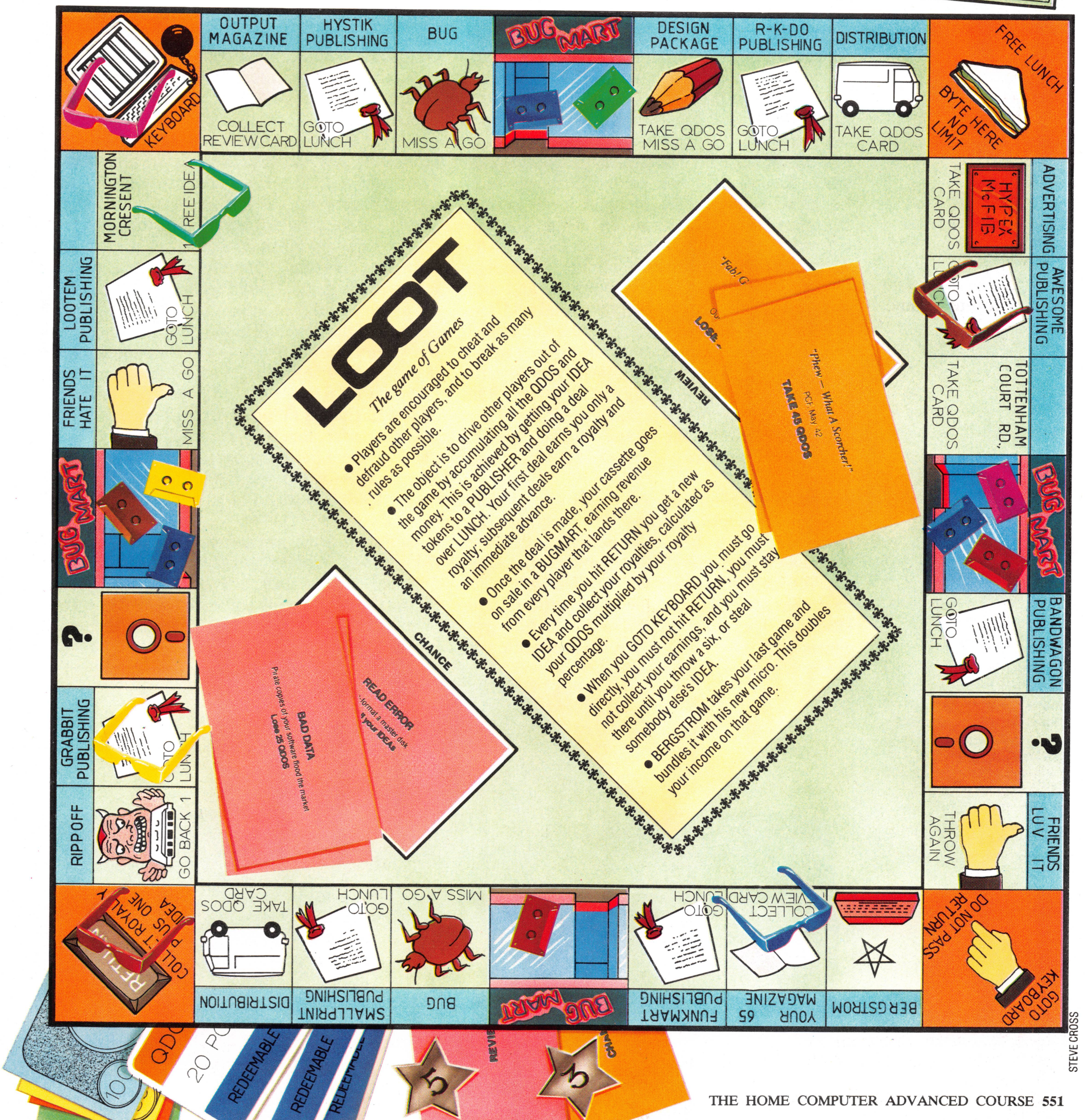
On a royalty deal, the programmer will retain copyright to the game. This means that if there is any dispute the programmer is the one who sues,

IAN McKINNELL

Retail Detail
Point of sale presentation is perhaps the most important of all user interfaces: here, the effects of advertising, reviews, packaging, merchandising and product appeal combine in influencing the customer's choice. Availability is obviously crucial to sales, so getting the product distributed through multiple outlets such as Boots and W H Smith is a primary goal of all software publishers

Making It
Thousands of home micro owners are trying to write software packages: some of them actually succeed, find a publisher and make some money. Our imaginary game may overstate the element of change in the process, but a lottery it surely is . . .





LOOT

The game of Games

- Players are encouraged to cheat and defraud other players, and to break as many rules as possible.
- The object is to drive other players out of the game by accumulating all the QDOS and money. This is achieved by getting your IDEA tokens to a PUBLISHER and doing a deal over LUNCH. Your first deal earns you only a royalty; subsequent deals earn a royalty and an immediate advance.
- Once the deal is made, your cassette goes on sale in a BUGMART, earning revenue from every player that lands there.
- Every time you hit RETURN you get a new IDEA and collect your royalties, calculated as your QDOS multiplied by your royalty percentage.
- When you GOTO KEYBOARD you must go directly, you must not hit RETURN, you must not collect your earnings, and you must stay there until you throw a six, or steal somebody else's IDEA.
- BERGSTROM takes your last game and bundles it with his new micro. This doubles your income on that game.

STEVE CROSS

or gets sued. If the programmer receives a single lump sum for his game, the software company will generally buy the copyright to the game too.

In addition to a royalty Harrap is also paid a retainer fee, but he is under exclusive contract to Gremlin Graphics for a year and cannot work for any other software house. Many contracts will also have an option clause written into them that will give the software company the exclusive right to first choice, or refusal, of any idea or program.

For a game to be successful, a company looks for sales of at least 15,000 during its lifetime, which typically is four or five months. In the case of Wanted: Monty Mole the company initially duplicated 10,000 copies for each machine (Spectrum and Commodore 64), with the option to duplicate more. When the game was launched Ian Stewart was predicting sales of 20,000 for each machine and said: 'I feel that it will definitely go to number one.'

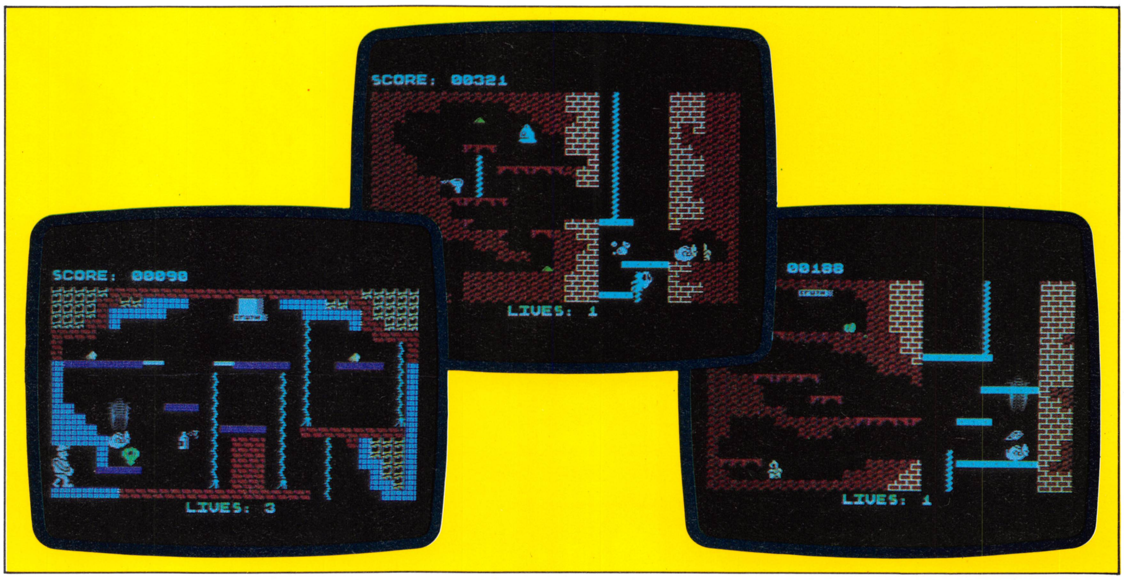
Christopher Kerry is another programmer for whom success came even earlier. At the age of 17 he left school before taking his A levels to work freelance for the House of Thor software company. Kerry wrote the game Jack and the Beanstalk on the Spectrum computer he had at his home in Sheffield. With no publicity and after being on sale for just two months, it had sold 'tens of thousands' according to the company (which was not prepared to reveal the exact figure). In fact the House of Thor was very protective towards Kerry, the reason given being that they were afraid of some other company 'poaching' him. As well as Christopher Kerry, the House of Thor uses 15 other programmers — most of whom are teenagers.

Like all software houses, the House of Thor welcomes people sending in programs for evaluation and if they look promising, then the company will contact the programmer that same day. A speedy response is vital as other companies might have been offered the same software. The House of Thor is only interested in arcade-type games and these can only really be written in machine code. The company is also looking for original themes and so will not consider programs that are near-copies of existing games: you cannot simply alter the graphics or change the names of monsters, for example. Apart from possible legal action, such games are unlikely to sell well.

Salamander is a larger software house with nearly 40 programmers on its books. Chris Holland, the Managing Director, gave some further hints to potential games writers: 'Where you can make a killing is by writing software for new machines when they've just come out. We welcome software for the Amstrad, the Atmos and the MSX micros. With something like the Spectrum it's more difficult unless it's a completely new idea.' The company has a range of about 50 titles for most of the popular home computers — not just the Commodore 64 and Spectrum.

Rather than trying to sell programs to software houses, a few programmers start their own. Llamasoft and Bug-Byte are just some of the software companies that have been founded by young games programmers. Unfortunately you do need quite a lot of capital to get started. Ian Stewart's company, Gremlin Graphics, started with a single game, Potty Pigeon, but are hoping for a range of five programs by the end of 1984. All four of the directors have had to put their own money into the company, to pay for publicity and distribution: a colour page advert in a computer magazine costs over £1,000 if production costs are included. Even a range of successful games does not guarantee financial security. Imagine (see page 79) went bankrupt in July 1984 owing over £1 million — yet at one point the company was selling £300,000 worth of games software every month.

Setting up your own software company can be risky: as well as good games you also need sound financial advice. But anyone can send in a program to an existing software house 'on spec'. Who knows, you might have written the next number one.



The Pits
The programmer who wrote Wanted: Monty Mole is the archetypal software whizz-kid — young, self-taught, bright and well-paid. The game itself is far from typical: it takes a flippantly anti-union look at the 1984 miners' strike, but every copy sold will gain the Miners' Welfare Fund a five pence donation from the publishers

LIZ HEANEY

WIRED FOR SOUND

In the course so far, we have seen how a musical digital interface is used by producers in the studio and by musicians performing live. Now we consider some of the hardware and software that puts the MIDI in the hands of the home computer user, taking a detailed look at packages representative of current technology.

The Micon, or *MI*DI *con*troller, is produced by XRI Systems, and is fairly typical of the MIDI packages produced by the smaller software houses. It is designed for use with the Sinclair Spectrum, and consists of a MIDI interface and a software cassette. Its sequencer facility provides for eight separate tracks, each with a capacity for 2,951 events (notes, rests, etc.). Music is entered on the interfaced synthesiser's keyboard, and the Spectrum's Space key is used to define the number of events to be played.

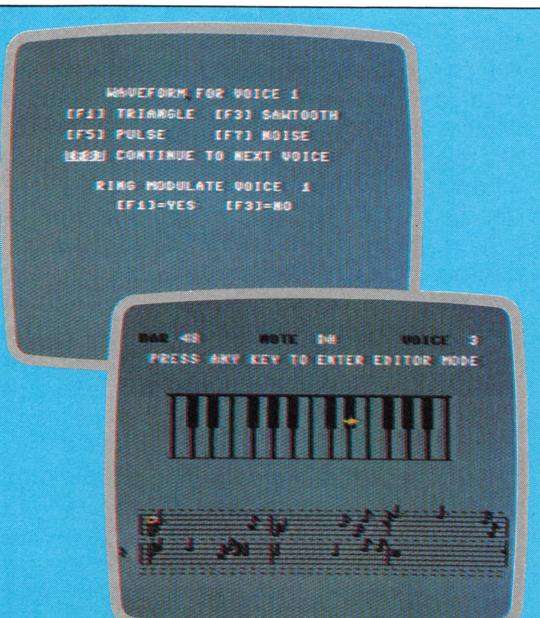
As the synthesiser is being used, the music rolls by on the screen in a crude form of standard five-line stave notation. This gives bass and treble clefs, the full range of durations (including dotted notes), sharps and flats, and staccato (very short note) indicators. The notation is poor on indicating rests — those durations where *no* note events occur, a crucial component of any musical composition — and the stems from the noteheads always point upwards, instead of up or down depending on their placement on the stave. In addition, sequences of short notes are not rationalised into groupings that reflect the metre or underlying pulse of the music. The result is hard to follow except as a rough guide, but it can be printed out using the ZX printer.

The main value of the display is evident when we come to editing the composition. The music is divided into bars, and, when a bar number is entered, individual notes can be deleted, inserted or altered by specifying the new note. Whole bars can be erased, and bar groupings can be repeated anywhere within the sequence. Up to 10 sequences — nearly 24,000 notes — can be saved on tape, together with the appropriate characteristics for replay. The tempo of the replay can be defined to within four milliseconds or controlled by a drum machine using a SYNC IN socket on the interface.

Micon also sequences in real time. In effect, it will 'listen' to a performance on a MIDI-compatible synthesiser, and enter all the data into the Spectrum's memory. It won't display a notation of the performance, but it is useful in two other ways. First, it enables keyboard musicians to hear a recording of their own performance, without having to worry about setting up input levels onto tape: its immediacy should be particularly valuable in music education. Secondly, providing the required music is 'under the fingers' of the performer, it alleviates much of the tedium of step-time sequencing. However, the performer still needs a metronome or drum machine pattern to play against in order to keep a steady tempo, as the sequence will otherwise be replayed with all the performer's timing errors. As with any MIDI system, the results obtained from Micon depend upon the design sophistication of the synthesiser it is interfaced to. Micon costs just over £100.

Jellinghaus Music System's MIDI package costs a little less than the Micon, but is designed for use with a wider range of machines: the Apple II, Commodore 64 and the Sinclair Spectrum. It is fundamentally similar to Micon, in that musical information is entered from the keyboard of the interfaced synthesiser. The main difference is that sequences are entered in real time only, and the screen display of the music is not in standard five-line stave notation.

Organising the music into bars is done by specifying a time-signature — this is simply the number of quavers or crotchets per group. Four time-signatures are available: three, four or five crotchets per bar ($\frac{3}{4}$, $\frac{4}{4}$ or $\frac{5}{4}$ time), or seven quavers ($\frac{7}{8}$) and, as with the Micon, performance tempo has to be co-ordinated by a separate metronome or drum machine. If the real-time performance



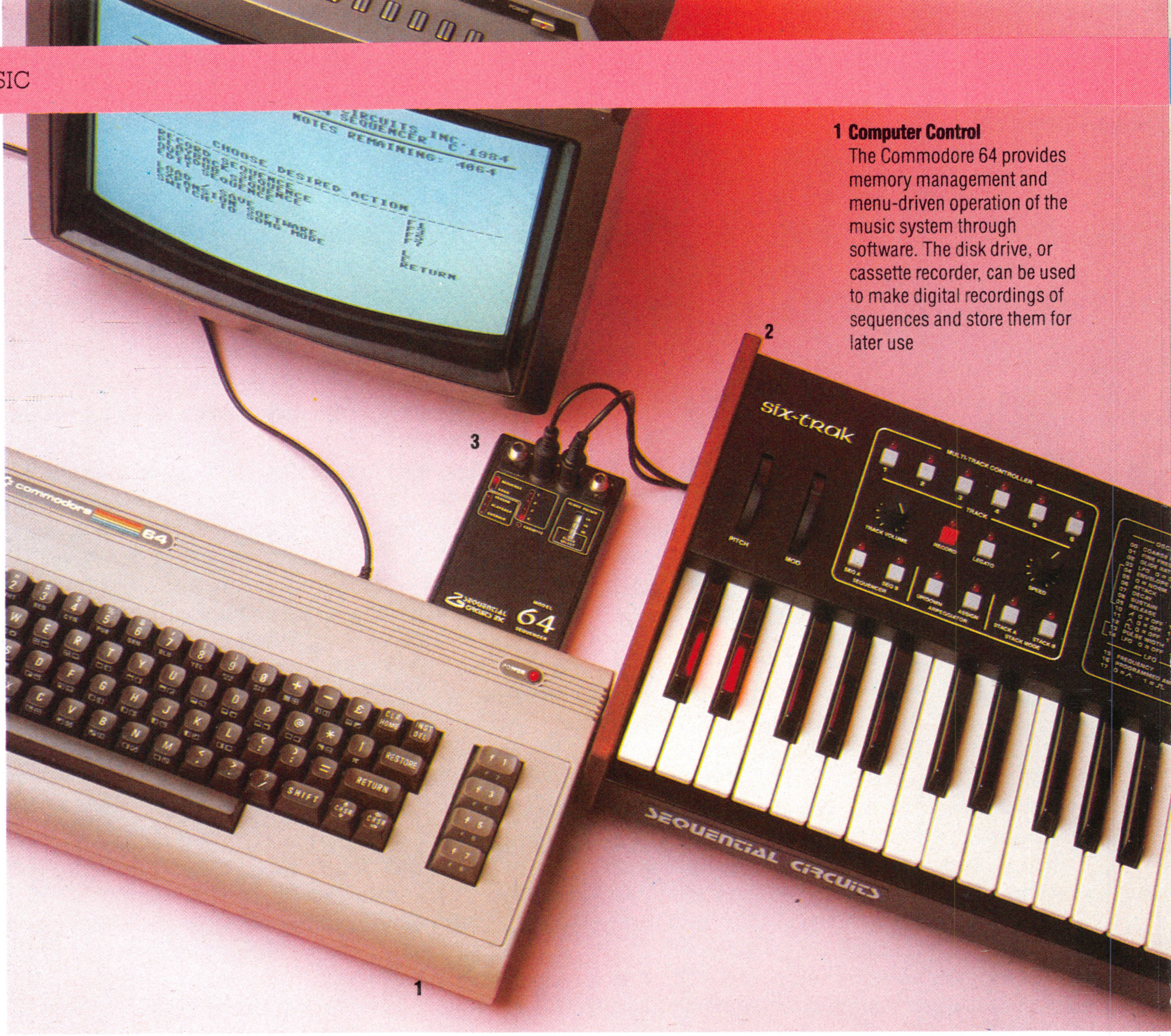
```
WAVEFORM FOR VOICE 1
[F1] TRIANGLE  [F3] SAWTOOTH
[F5] PULSE     [F7] NOISE
[F8] CONTINUE TO NEXT VOICE

RING MODULATE VOICE 1
[F1]=YES  [F3]=NO
```

```
BAR 48        NOTE D#        VOICE 3
PRESS ANY KEY TO ENTER EDITOR MODE
```

Sound Control

By taking advantage of the Commodore 64's sound chip, a BASIC program can be written to enter music in a visual format with a joystick, and have it played back. In the example shown here, a light appears on a graphic keyboard and can be moved up the scale by pressing forward on the joystick, or down the scale by pulling back. When you reach the desired note, you move the joystick from side to side to choose the duration of the note. The note is 'recorded' when the fire button is pressed, and you are ready to enter the next note in a sequence. (For a complete listing of this program, see 'Keyboard Capers' in Issue 3 of 'Your 64')

LIZ HEANEY



1 Computer Control
The Commodore 64 provides memory management and menu-driven operation of the music system through software. The disk drive, or cassette recorder, can be used to make digital recordings of sequences and store them for later use

deviates in tempo, then the input data will become a chain of 'un-barred' notes. Once the sequence is set to replay, there will be no problems until a grouping occurs that is neither $\frac{3}{4}$, $\frac{4}{4}$, $\frac{5}{4}$ nor $\frac{7}{8}$. As soon as this happens, the sequencer facility fails. This also means that a metre such as $\frac{12}{8}$ cannot be entered at all, even if the tempo is perfect: an unfortunate omission, since $\frac{12}{8}$ is a commonly used 'swing' metre in jazz, rock, funk and reggae.

Providing a sequence *has* been entered successfully, the data is displayed on the screen in columns, showing the octave in which the note-event occurs; the pitch of the note within the octave; the duration in musical terms (crotchets and quavers); 'gate'-time (a parameter used to add a sense of phrasing to the musical line), and the 'touch-sensitive' values of the keyboard, from 0-9.

This type of display in fact gives more information on a note-for-note basis than the Micon's use of standard notation, which has never incorporated either gate-timings or velocity levels — at least not in numerical form. This is because the ability to specify such parameters has entered the musical vocabulary only with the advent of electronics.

However, many musicians without formal training soon develop an ability to look at a section of standard notation and, by using the rise-and-fall shape of the notes outlined on the stave, get some idea of how a tune will sound, or what a group of notes in a chord adds up to harmonically. This means that, with practice, notation can be read, and a musical sequence 'heard' mentally in real time. Column displays may be excellent for checking data, but few people can hum a tune from them. An ideal system would include both types of display but, failing this, the Micon's crude version of standard notation is more effective and better suited to developing musical skills.

CONCEPTUAL SYNTHESISERS

The main problem facing the musically inclined home computer owner may not be that of finding the right MIDI package; it is likely to be obtaining a MIDI-compatible synthesiser sophisticated enough to make the best use of the interface without the user needing a second mortgage to pay for it. If we require that the synthesiser should be able to handle 50 per cent of the backing tracks for a typical 1984 pop single, we find that such instruments cost in the region of £800 to £1,500. Below that range, most synthesisers are extremely limited in their application.

A possibility to consider is what has been called the 'conceptual synthesiser' — typically, an advanced software package and keyboard peripheral that utilises the sound-generating potential of a microcomputer, rather than an interface to a synthesiser that is likely to have been designed before MIDI was developed.

An example is the PDSG (programmable digital sound generator), manufactured by Clef Products. This interfaces with the BBC Model B microcomputer. It includes a 61-note velocity-

2 Sequential Circuits' Six-Trak
This is a powerful synthesiser with a built-in multi-track tape recorder. The Six-Trak is so named because it can record on six different tracks. Unlike many such synthesisers, the Six-Trak allows for each musical voice to have a different sound which makes it possible to create complex ensemble sounds. Each voice can be controlled for frequency, timbre, waveform, frequency glide and note bending. The Six-Trak costs £795

3 Sequential Circuits' Model 64
By plugging the Model 64 into the memory expansion port of a Commodore 64, it is possible to incorporate a MIDI-equipped synthesiser and the computer's memory, cassette or disk storage and video display into a music system. The Model 64 stores timing, pitch and modulation information for up to 4,000 notes in record mode. For playing back, the interface can either send the digital signal exactly as it was received from the keyboard (real-time), or can correct it to a given time signature (step time). The Model 64 costs £185

IAN McKINNELL

sensitive music keyboard and two foot-pedals. The sensitive keyboard provides the difference between a responsive or a 'dead' touch in performance, and the sensitivity data from real-time playing can be stored for recording and replay. It has 32 sound-generating units, rather than oscillators, each of which can have up to 11 defined characteristics. If required, all 32 units can be used to produce a single note. This facility alone, in the hands of a capable user, gives a PDSG a richness and variety of sound on a par with most of the synthesisers we have listed in the box.

If a single generating unit is assigned to each note, then a programmed sequence can be made up of 32 individual lines. Alternatively, a proportion of generators can be used for sequenced material, and the remainder played in real time against the sequence. Waveform characteristics are displayed on the screen, giving the opportunity to analyse sounds visually — an invaluable back-up to aural guesswork, and a factor that makes the PDSG ideally suited to music education. The sound-generation package alone — adequate for non-real-time sequencing, waveform creation and analysis — costs about £200, with the keyboard at around the same price.

The main drawback to the PDSG is its representation of sound at the digital-to-analogue conversion stage. The human ear and brain can interpret sounds across a bandwidth from 20Hz to 20KHz. Natural sounds, including those produced by acoustic musical instruments, are active within the whole of this bandwidth and more. The PDSG, however, can represent sound only within a bandwidth of up to 12KHz. As a result, its sound quality is on a par with an adequate home hi-fi system, and the manufacturers assume that a home hi-fi amplifier and speakers will be used to complete the system. Most synthesiser players would be dismayed if this was their only choice of amplification. Clef Products plan versions of the PDSG to interface with other microcomputers.

MIDI has been seen as a breakthrough because it gives microcomputer owners access to real music synthesisers. The PDSG system is advanced enough in many respects for synthesiser owners to consider buying a microcomputer and a 'conceptual synthesiser' instead.

A Feel For The Music

The development of the MIDI interface provides microcomputer owners with a range of possibilities in music-making. But, at the same time, there is a risk of buying an expensive package — the interface itself, additional software and a synthesiser — only to be swamped by the intricacies of the system.

One alternative is to start with an inexpensive music system to become familiar with the basics of electronic music. Such a system, of course, must be good enough to be musically satisfying and stimulate an interest in the further possibilities of electronic music. A good 'starter package' is the cassette-based Ultisynth 64, produced by Quicksilva. This exploits the Commodore 64 SID (sound interface device) chip and its three oscillators.

Using the package, each key of the Commodore's keyboard becomes an independent control for generating and defining sound. The four basic wave shapes — sine, square, triangle and sawtooth — are available, together with incremental settings for defining the attack-sustain-decay-release (ADSR or envelope) characteristics. Sounds can be filtered (i.e. a specified bandwidth of frequencies can be subtracted from the output) to characterise the sound still further. Ring modulation — a process that gives the sum and difference of any two frequencies — is also included. This is useful for producing quite authentic bell-like sounds. In addition, rhythm can be written in, pre-set rhythms incorporated, and 2,048 notes can be sequenced.

The Ultisynth facilities closely resemble those of the VCS 3, a 'classic' voltage-controlled synthesiser of the late 1960s and early 1970s. Today, the Ultisynth package costs only £14.95, while the VCS 3 is acquiring the status of a museum piece.

Another cassette-based system suitable for the beginner is Romik's Multisound, which is very similar to Ultisynth in its control facilities, but gives a graphic display of a music keyboard. Positions on this keyboard are selected using a cursor and the notes defined by information entered on the computer's alphanumeric keyboard. This makes the package more 'musician-friendly', exploiting any previous familiarity the user may have with a music keyboard.

CHOICE OPTIONS

In any complex program there will be points at which the user will need to branch to one of a number of available options. The menu system lays out the list of choices at certain specified stages for the user to select from; a command system allows the user to choose from a range of options at any time. We consider both techniques.

Menus may be simple lists of numbered items or they may be screens full of elaborate icons, but the principle behind their use is the same. A menu is used when the program reaches a multi-way branch in its logic; the user is asked to choose which route to take from a list of available options displayed on the screen. Menu-based programs tend to be tree structures: the user enters the tree at the 'root' and is guided by the menu options towards one of the 'leaves', where the information or function is to be found.

The major advantage of this approach is that the user needs little or no knowledge of the program structure because the route is well 'signposted' all the way. However, more experienced users find the job of navigating their way through a chain of menus tedious for often-repeated tasks. Novices, too, may have difficulty with the tree structure; correcting a wrong decision involves working back through all the menus to the point at which the mistake was made, re-entering the correct option, and then continuing from there. Prestel is a particularly 'deep' structure of this type, and users frequently encounter this problem. Menus need not form a tree at all — they may be organised into a network by using loops. However, this tends to increase the risk of getting lost within the program structure and so is not suitable for novice users.

Designing a menu system can be difficult, although the actual programming is relatively easy. The main problem is that the entire program must be clearly specified before any code is written. (This is good practice anyway, but is not always straightforward.) Adding new functions at a late stage can involve changing several menus earlier in the program, and this may require major restructuring. When the program is designed, all menu logic should be included in a single routine that calls the routines at the 'leaves' when these are reached. The menu routine can thus be seen as a more complex form of the normal control routine, with all internal branching controlled by the user. This keeps the design tidy, and serves to separate the control logic from the functional parts of the program, allowing each to be developed and debugged independently.

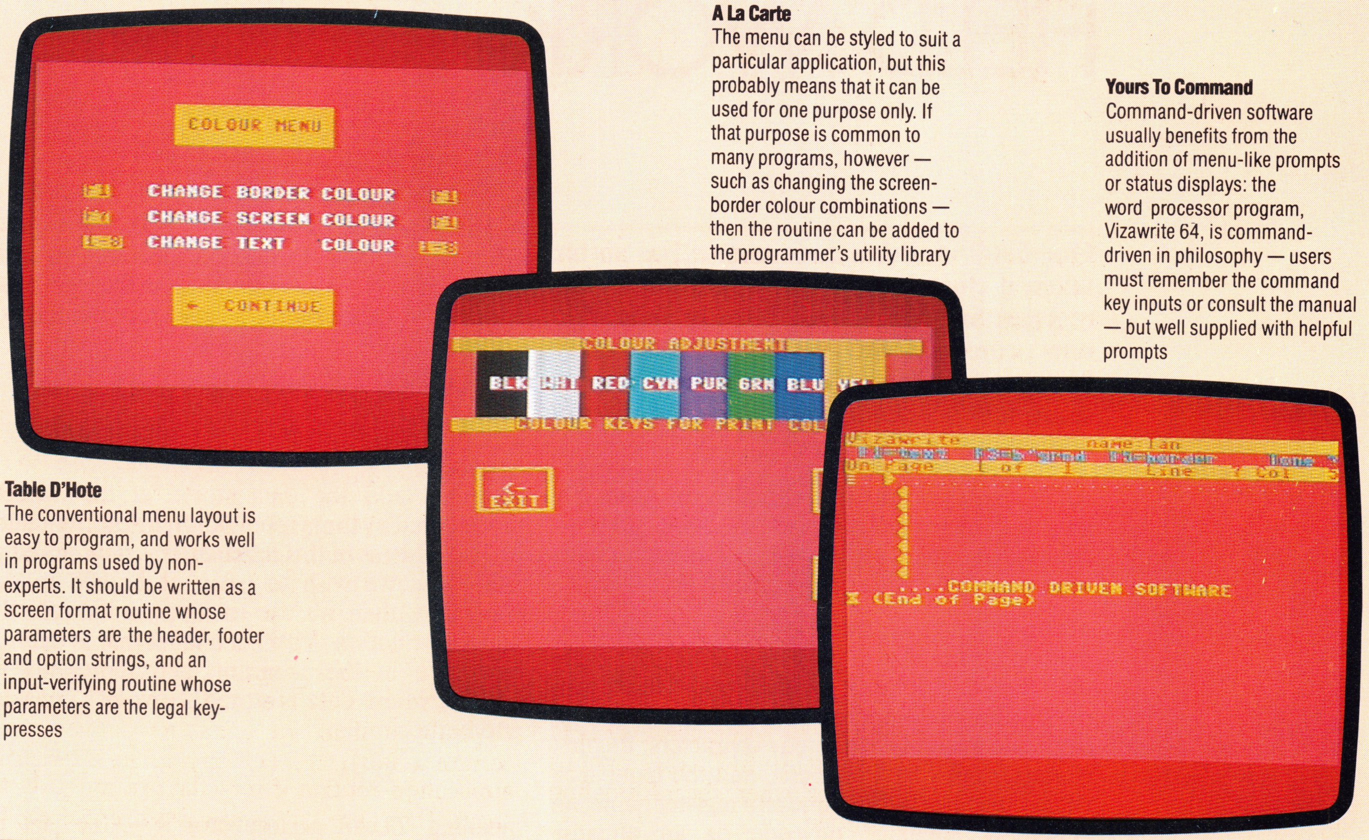
Program flow will now follow a set pattern. For each menu along the route, the menu logic routine passes a set of user prompts to a routine that puts them into 'slots' in a menu 'frame'. The slots will probably contain a screen header message that displays a title and any other necessary information about the menu, a 'footer' that explains how to make the choice (with sufficient space for the user's response), and the menu options themselves. In general, the most effective menu layout has up to eight options displayed in a column, with the response code (number, letter, mnemonic, etc.) at the left of each item.

The menu routine calls an input routine, perhaps passing to it the conditions that must be specified for a legal input, and accepts in return the user's response. It then interprets this response (typically a single keypress) and either passes control to the next menu or calls the appropriate application routine if it is the last menu in the chain. Once the routine has been executed, the menu from which it was called might be redisplayed, or control could pass to some other part of the program (the root menu, perhaps).

Menus require a lot of text for headers, footers and prompts, but much of this will be repeated for each menu 'frame'. The explanation of how to choose a menu option (the 'help' command), an option offering an exit to the root menu, and other recurring choices may all be required by several different menus. If this is the case, space may be saved and the logic made clearer if all prompts are held in a string array (or on a random access disk file), from which they may be called by their index number. Design the menu display routine to accept references to this array and to display the appropriate headers, footers, prompts, etc.

A command-driven system is one that has a range of commands available to the user at any stage in the program. Each command goes straight to a subroutine that performs the required function. This system must be designed to inspect all input to ascertain whether it is data or a program command. The difference is usually signalled by the user pressing a particular key before each command input. The Control key is often used for this purpose. A word processor, for example, might accept the word 'save' as just one more word of text, but interpret it as a storage command if the Control key was pressed before the word was typed.

In a command-driven system, the 'tree' is very shallow and broad, and a single routine, acting as the control program, is used to direct the user to the required subroutine. This 'command interpreter' has four main tasks. The first is simply


COLOUR MENU
F1 CHANGE BORDER COLOUR F2
F3 CHANGE SCREEN COLOUR F4
F5 CHANGE TEXT COLOUR F6
← CONTINUE
COLOUR ADJUSTMENT
BLK WHT RED CYN PUR GRN BLU YEL
COLOUR KEYS FOR PRINT COL
<- EXIT
Vizawrite name Ian
F1=text F3=b'grnd F5=border
On Page 1 of 1 Line Col
....COMMAND DRIVEN SOFTWARE
(End of Page)

A La Carte
The menu can be styled to suit a particular application, but this probably means that it can be used for one purpose only. If that purpose is common to many programs, however — such as changing the screen-border colour combinations — then the routine can be added to the programmer's utility library

Yours To Command
Command-driven software usually benefits from the addition of menu-like prompts or status displays: the word processor program, Vizawrite 64, is command-driven in philosophy — users must remember the command key inputs or consult the manual — but well supplied with helpful prompts

Table D'Hote
The conventional menu layout is easy to program, and works well in programs used by non-experts. It should be written as a screen format routine whose parameters are the header, footer and option strings, and an input-verifying routine whose parameters are the legal key-presses

IAN McKINNELL

Two Into Three
There are usually many ways of solving the same problem — these displays show different ways of allowing the user to change the screen, border and text colours on a Commodore 64

to wait for an input. The second is to 'parse' this input — the interpreter must separate the input line into its functional units. The third task is to interpret the command by preparing the appropriate subroutine call. (What is the routine's address? Are there parameters to be passed?) Finally, it must actually call the routine to be executed. When control returns, the interpreter goes back to its first job — waiting!

The format of a command may be extremely elaborate, and some command languages are similar to a simplified form of English. An example of a command language is the Unix shell, where the typical command format is:

```
Command + optional parameter list
e.g. L
or L -1
```

Here, the Unix command L lists a file directory, while L -1 (where -1 is an optional parameter) lists a file directory in 'long' format.

The parser must be able to recognise the various parts of the command line. Unix keeps things simple (in most cases) by taking the first word as the command and recognising parameters by a preceding minus sign. Command language parameters are not for the use of the command interpreter itself, but are required by the subroutines that the interpreter calls. Routines used in the command system should ideally adopt a standard format for input parameters. If this is done, the command interpreter can pass the parameters in the form in which they were entered (as strings, perhaps).

It is obviously much easier to create a command interpreter than to write a menu system. Experienced users tend to prefer command systems, as these are faster and more flexible than menu-driven programs. Most operating systems are command driven, which is unfortunate for novice users as such systems provide no signposting facilities and the on-line help routines (if there are any) require some knowledge of the system. In addition, the sheer number of commands and optional parameters in a typical command system means that even those reasonably familiar with the system will require help facilities or need to consult the operating manual frequently.

Beginners hate commands and experts hate menus. This problem is virtually insoluble, although some hybrid systems exist that can be quite effective. For example, the Wordstar word processing program is basically a command-driven system, but it can appear to the user to be a menu system. The commands are control codes (some with parameters) and the user runs the system entirely with these. The menus that appear on the screen use these commands as mnemonics for selecting options so that, as the novice uses the menus to run the program, the commands are learnt at the same time. The help level may be set to permit the menus to be dispensed with once the user is sufficiently proficient to find them a nuisance. The Wordstar menu system is only two levels deep, however; this approach would be more difficult to implement with more complex systems.

FIELD WORK

Our 6809 machine code course has so far taken a close look at the operation of the registers in storing and moving data. We now consider the layout of Assembly language programs in more detail, inspect some new assembler directives and discuss addressing modes.

As it is entered into an assembler, an Assembly language statement consists of three parts — although not all of these need be present for each statement. The three parts or *fields* are:

- **The Label Field,** which is found in the leftmost column of the screen display. Labels are simply identifiers representing numbers, which are usually memory addresses. A label consists of from one to six alphanumeric characters: the first character must be alphabetic, and the name as a whole must not be the same as a register name, an Assembly language op-code or an already-defined label. These are the usual assembler conventions, but they are only conventions — different assembler programs may follow different rules. If you do not wish to label a line, then you should start it with at least one space character, indicating an empty label field. Similarly, a space terminates the label field; LABLDA, for example, is a valid label, whereas LAB LDA is interpreted as the label LAB followed by LDA, the op-code.

The assembler maintains a *location counter,* which is the equivalent of the processor's program counter register. It holds the address of the memory location where the next byte of instruction or data is to be stored. When the assembler first encounters a label, it saves the identifier in an area of memory called the *symbol table* — this is like an array in a BASIC program. Along with the identifier it stores the address held in the location counter at the point when the label was first encountered. Whenever the assembler encounters a label in the Assembly language program, it inspects the symbol table for that label. If it is in the table then the assembler replaces it by the address given for it, and if it is not in the table then the assembler stores it there along with the location counter contents.

- **The Operator, Instruction or Op-code Field** is found to the right of the Label Field. This is a mnemonic, usually consisting of three characters, and — if necessary — a register name. Thus, ADDA is formed from the mnemonic ADD and the register name A. The op-code represents the processor operation to be executed. Like the label field, it is terminated by space.

- **The Operand or Address Field,** which gives information about the data on which the op-code is to operate. Normally this data is in the form of an address, or more often, a label representing an address.

Consider this Assembly language statement:

```
LABEL1   ADDA   NUM1
```

meaning 'add the data stored at the address represented by the symbol NUM1 into accumulator A'. The address of this instruction is now stored as LABEL1; if we wish to jump or branch to this statement, then we use its label to indicate the jump destination. NUM1 is a label that is defined elsewhere in the program, and represents an address where data is stored. Now let's consider another example:

```
         CLRA
```

meaning 'CLeaR accumulator A — i.e. set its contents to zero'. This is an example of an op-code that needs no operand. Notice that we have not labelled this line. Labels are entirely optional — you should use them because the line will be branched to by some other instruction, or as a kind of REM statement, labelling significant lines with explanatory remarks.

Using labels in this second way is helpful, but it is no substitute for full explanatory comments. These can be added to any line by leaving a space after the last character of the operand, and then inserting your comment. Some assemblers may require a special character to indicate the beginning of a comment, and it is normal to start a whole new line of comment this way, usually with an asterisk.

Constants, in the form of numbers or character strings, can be used in the operand field. Numbers are usually taken as being in decimal, unless indicated as hexadecimal by a $ prefix or an H suffix (e.g. $AF08, AF08H); or octal (number base eight) by a @ prefix or a Q suffix (e.g. @6712, 6712Q); or binary by a % prefix or a B suffix (e.g. %11010011, 11010011B). The ASCII code of a character can be used as an operand by prefixing it with an apostrophe, thus 'A, meaning 65 or $41.

A particularly useful value is the current location counter contents. This is not usually known to you when you enter the program, but can be referred to in the operand field by an asterisk. Most assemblers will accept this in simple arithmetic expressions — usually restricted to arithmetic and subtraction. For example:

```
         LDA     *+5
```

means 'load the accumulator with the contents of the memory location whose address is five bytes higher than the present contents of the location counter'.

ASSEMBLER DIRECTIVES

An assembler will normally accept a number of *directives* or *pseudo-ops*, which are used in the program like ordinary op-codes. We have already used two of them (see page 538):

```
CR        FCB     13
```

(Fix Constant Byte) reserves a single byte at the current location counter address, and gives it the value of the operand — here, the location whose address is represented by the symbol CR is initialised with 13.

```
MEMTOP    FDB     $7FFF
```

(Fix Double Byte) does the same for a two-byte (16-bit) value. Memory space can be reserved in blocks without fixing the value of its contents by using the RMB pseudo-op, as in this example:

```
TABLE1    RMB     7
TABLE2    FCB     $F6
```

which reserves seven bytes for a table of values whose first byte has the address represented by the label TABLE1. In practice, this means that if TABLE1 represents the address $C104, say, then TABLE2 will represent an address seven bytes higher — that is, $C10B.

An entire character string can be placed in memory, like this:

```
ERRMSG    FCC     'ERROR
```

This initialises five bytes of memory with the ASCII codes for E,R,R,O and R. Consequently, it is much easier to include user messages and prompts in Assembly language programs.

An important pseudo-op is ORG (ORiGin), which specifies a value for the location counter, and is used at the start of a block of program to instruct the assembler where in memory to begin locating the program when it is translated into machine code. You should always start programs with an ORG statement, though some assemblers may supply a default value if you don't. It is permissible, and sometimes desirable, to have more than one ORG statement in a program. An ORG directive does not take a label or an operand.

Perhaps the only statement that you might want to put before the ORG is EQU (EQUals or EQUate), since it specifies variable symbols and does not refer to the location counter. For example:

```
RESET     EQU     $F100
```

means that the symbol RESET is defined as representing the value $F100. RESET is thus shorthand for a number, whereas a label placed at the start of a statement line represents the address where data or machine code is stored.

Appropriately, the last directive we need consider is END, which is placed at the end of the source code as a terminator for the assembler. Like ORG, it takes neither a label nor an operand.

6809 ADDRESSING MODES

A measure of any Assembly language's power is its addressing modes — that is, the number of ways that the operand can be interpreted. The 6809 processor supports many such modes. The sample instructions that we have seen so far have all used either *direct* or *extended* mode, meaning that the value or label in the operand field is the address of the memory location that contains the data.

Direct addressing means specifying only a single byte address for the instruction operand. This is treated by the processor as the lo-byte of the full two-byte operand address, and it takes as the hi-byte of the address the contents of the *direct page register*, an eight-bit CPU register addressable by the program. The advantage of this mode is flexibility and generality: a subroutine, for example, can be written using direct addressing, and, therefore, not refer explicitly to any particular area of memory. Specifying the direct page register contents before calling the routine would 'point' it wherever the data lay in memory.

Extended addressing means specifying a two-byte address as the instruction operand. This instruction will then always refer to that particular byte of memory, and so is an inflexible piece of code. The assembler recognises direct and extended modes by the nature of the operand.

Another commonly-used mode is *immediate*, where the actual data is contained in the operand field itself. This mode is indicated by prefixing the operand with a #, the hash symbol. For example:

ORG $1000	start address of the machine code is $1000
NUM1 FDB $FFFF	initialises location NUM1 with $FFFF
LDD NUM1	loads $FFFF, the contents of NUM1, into the D register
LDD #NUM1	loads $1000, the actual (or immediate) value of NUM1, into the D register

Notice that the label NUM1 represents the address $1000. You might expect that the ORG instruction would reside at address $1000, and that the following instruction (and, therefore, its label — NUM1) would have a higher address. Remember, however, that ORG is an instruction to the assembler program about translating the Assembly language program, and not a part of the program itself. It does not, therefore, take up any memory space, so NUM1 takes the value $1000 because that is the value of the location counter when NUM1 is first encountered by the assembler. You should also notice that LDD NUM1 loads the *contents* of location NUM1 ($FFFF as specified by the FDB directive) into the D accumulator, whereas LDD #NUM1 loads the *value* of NUM1 itself (the address $1000 of the label).

SMALL UNDERTAKINGS

Casio is perhaps best known in the UK for the production of watches, calculators and musical instruments. Over the last few years, however, the company has introduced a range of hand-held and pocket computers in an attempt to gain entry to this specialist computer market.

Casio claims to control 50 per cent of the world calculator market but, surprisingly, the company has just 3,300 employees worldwide. In 1983 Casio's turnover was $29 million — a low figure for a high-volume electronics manufacturer. Tony Manton, the UK calculator sales manager, states that this is a deliberate approach. 'We are a very small company and pretty conservative. Huge advertising campaigns aren't our style,' he says.

The company was founded by the five Kashio brothers after the Second World War. It was then known as Kashio Seisakujo, and started by manufacturing office equipment. In the early 1950s, the company developed the 14-A Relay Calculator; this was one of the first electrical calculating machines and was as big as a desk top, weighing 130kg (286 lb). Further calculators were developed, and in 1957 the company name was changed to Casio Computer Company Ltd.

Casio UK was established in 1974, and found a ready market for low-priced electronic calculators. In 1982 the first Casio pocket computer, the FX-702P, was launched. This was designed for the scientific user and had an unusual keyboard layout, with the keys arranged in alphabetical order instead of the more usual QWERTY format. Sales were disappointing, and the machine was replaced by the FX-700P, which was equipped with a QWERTY keyboard.

This was followed by the PB-100, a pocket-sized computer aimed at the business user. Similar in design to the FX-702P, it featured a liquid crystal display and numeric keypad. Casio also produced custom-built cassette recorders, printer/plotters and RAM packs for both machines.

A recent replacement for the PB-100 is the FX-750P. This computer features two 'RAMcard' slots, which take small metal packs (about the size of a book of matches) that can hold up to 4 Kbytes of data. Each card is fitted with a three-volt battery that stores the memory contents once the card has been removed from the computer, thus allowing programs to be saved and loaded into the machine as required. Each battery gives a year's storage; when the battery needs to be replaced the programs are reloaded from tape.

Casio also produces the FP-200, a hand-held computer that is equipped with 8 Kbytes of RAM. This can be expanded to 32 Kbytes. The FP-200 has an LCD display, giving eight rows of 20 characters in text mode and supporting a graphics resolution of 160 × 64 pixels. The latest machine in the Casio range is the SL-800. This is a low-priced calculator that is about the size and weight of a credit card. The slim design is a result of a manufacturing process known as 'filming', in which components are printed onto laminated film rather than being soldered onto a conventional circuit board.

In many European countries and the Far East, Casio distributes business computers and MSX standard home computers. In the UK, however, the company has concentrated on calculators and small computers. Asked why Casio has not attempted to break into the British home and business market, Tony Manton cites the cut-throat nature of this sector of the industry. 'We intend to expand upwards slowly,' he says. 'We need to educate people to tell them that pocket and hand-held computers are more than calculators.'

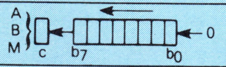
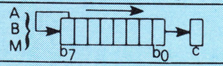
In addition to any future expansion in this area, Casio is using the expertise gained in the electronic keyboard field to produce a series of machines based around the MIDI digital interface (see page 534). Such machines should make their market debut by the end of this year.

Casio Research and Development Centre, Tokyo

Top Of The Tree
Tadao Kashio, President of Casio, is one of five members of the Kashio family at the top of the Casio corporate structure

DATABASE

Here, courtesy of Motorola Inc, the manufacturers of the 6809, we reproduce the first instalment of the 6809 programmer's reference card.

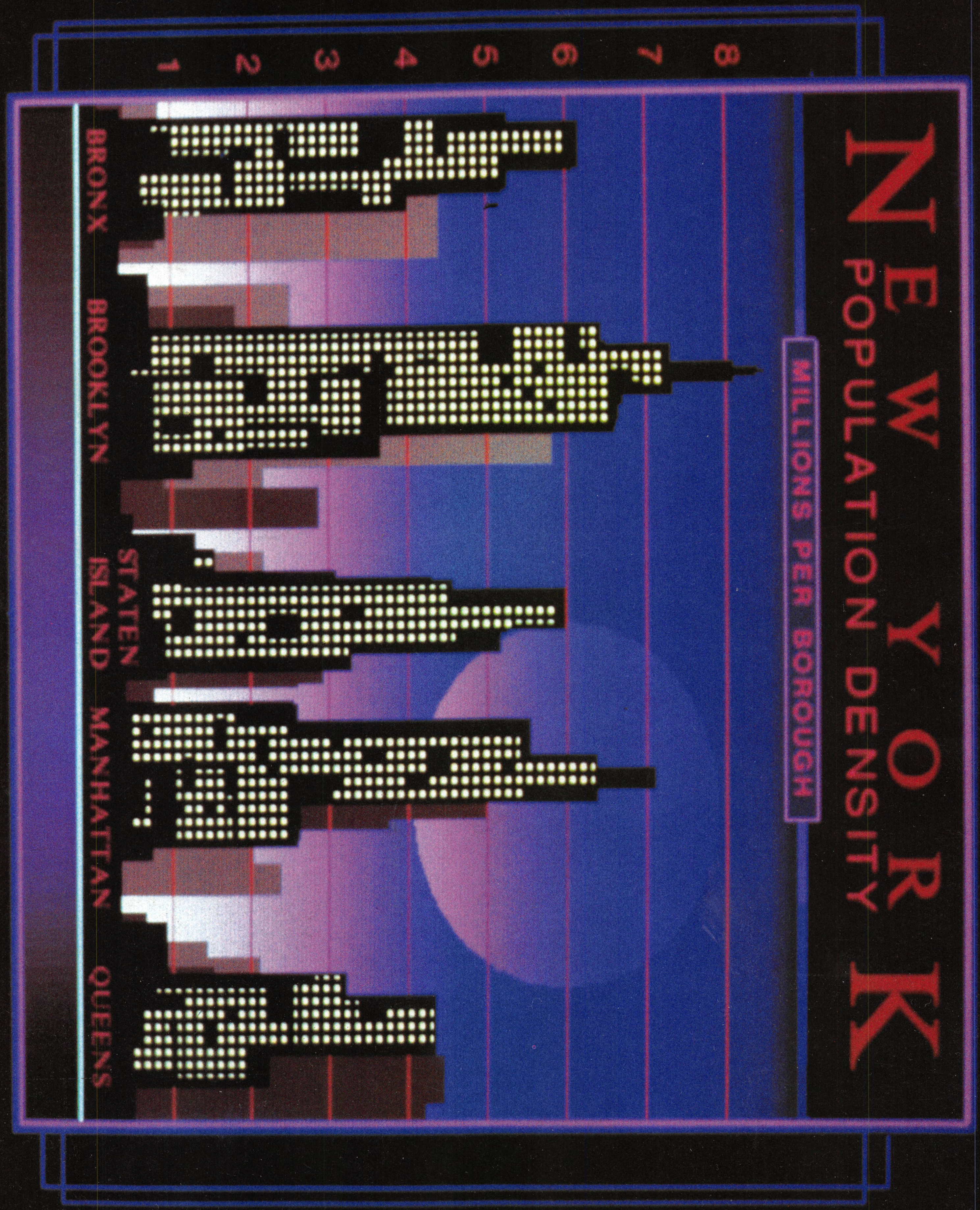
Instruction	Forms	Addressing Modes															Description	5	3	2	1	0
		Immediate			Direct			Indexed			Extended			Inherent								
		Op	~	#	Op	~	#	Op	~	#	Op	~	#	Op	~	#		H	N	Z	V	C
ABX														3A	3	1	B + X → X (Unsigned)	•	•	•	•	•
ADC	ADCA	89	2	2	99	4	2	A9	4 +	2 +	B9	5	3				A + M + C → A	↕	↕	↕	↕	↕
	ADCB	C9	2	2	D9	4	2	E9	4 +	2 +	F9	5	3				B + M + C → B	↕	↕	↕	↕	↕
ADD	ADDA	8B	2	2	9B	4	2	AB	4 +	2 +	BB	5	3				A + M → A	↕	↕	↕	↕	↕
	ADDB	CB	2	2	DB	4	2	EB	4 +	2 +	FB	5	3				B + M → B	↕	↕	↕	↕	↕
	ADDD	C3	4	3	D3	6	2	E3	6 +	2 +	F3	7	3				D + M:M + 1 → D	•	↕	↕	↕	↕
AND	ANDA	84	2	2	94	4	2	A4	4 +	2 +	B4	5	3				A ∧ M → A	•	↕	↕	0	•
	ANDB	C4	2	2	D4	4	2	E4	4 +	2 +	F4	5	3				B ∧ M → B	•	↕	↕	0	•
	ANDCC	1C	3	2													CC ∧ IMM → CC					7
ASL	ASLA													48	2	1		8	↕	↕	↕	↕
	ASLB													58	2	1		8	↕	↕	↕	↕
	ASL				08	6	2	68	6 +	2 +	78	7	3					8	↕	↕	↕	↕
ASR	ASRB													47	2	1		8	↕	↕	•	↕
	ASR													57	2	1		8	↕	↕	•	↕
	ASR				07	6	2	67	6 +	2 +	77	7	3					8	↕	↕	•	↕
BIT	BITA	85	2	2	95	4	2	A5	4 +	2 +	B5	5	3				Bit Test A (M ∧ A)	•	↕	↕	0	•
	BITB	C5	2	2	D5	4	2	E5	4 +	2 +	F5	5	3				Bit Test B (M ∧ B)	•	↕	↕	0	•
CLR	CLRA													4F	2	1	0 → A	•	0	1	0	0
	CLRB													5F	2	1	0 → B	•	0	1	0	0
	CLR				0F	6	2	6F	6 +	2 +	7F	7	3				0 → M	•	0	1	0	0
CMP	CMPA	81	2	2	91	4	2	A1	4 +	2 +	B1	5	3				Compare M from A	8	↕	↕	↕	↕
	CMPB	C1	2	2	D1	4	2	E1	4 +	2 +	F1	5	3				Compare M from B	8	↕	↕	↕	↕
	CMPD	10	5	4	10	7	3	10	7 +	3 +	10	8	4				Compare M:M + 1 from D	•	↕	↕	↕	↕
		83			93			A3			B3											
	CMPS	11	5	4	11	7	3	11	7 +	3 +	11	8	4				Compare M:M + 1 from S	•	↕	↕	↕	↕
		8C			9C			AC			BC											
	CMPU	11	5	4	11	7	3	11	7 +	3 +	11	8	4				Compare M:M + 1 from U	•	↕	↕	↕	↕
		83			93			A3			B3											
	CMPX	8C	4	3	9C	6	2	AC	6 +	2 +	BC	7	3				Compare M:M + 1 from X	•	↕	↕	↕	↕
	CMPY	10	5	4	10	7	3	10	7 +	3 +	10	8	4				Compare M:M + 1 from Y	•	↕	↕	↕	↕
		8C			9C			AC			BC											
COM	COMA													43	2	1	$\overline{A}$ → A	•	↕	↕	0	1
	COMB													53	2	1	$\overline{B}$ → B	•	↕	↕	0	1
	COM				03	6	2	63	6 +	2 +	73	7	3				$\overline{M}$ → M	•	↕	↕	0	1
CWAI		3C	≥20	2													CC ∧ IMM → CC Wait for Interrupt					7
DAA														19	2	1	Decimal Adjust A	•	↕	↕	0	↕
DEC	DECA													4A	2	1	A − 1 → A	•	↕	↕	↕	•
	DECB													5A	2	1	B − 1 → B	•	↕	↕	↕	•
	DEC				0A	6	2	6A	6 +	2 +	7A	7	3				M − 1 → M	•	↕	↕	↕	•
EOR	EORA	88	2	2	98	4	2	A8	4 +	2 +	B8	5	3				A ⊻ M → A	•	↕	↕	0	•
	EORB	C8	2	2	D8	4	2	E8	4 +	2 +	F8	5	3				B ⊻ M → B	•	↕	↕	0	•
EXG	R1, R2	1E	8	2													R1 → R2[2]	•	•	•	•	•
INC	INCA													4C	2	1	A + 1 → A	•	↕	↕	↕	•
	INCB													5C	2	1	B + 1 → B	•	↕	↕	↕	•
	INC				0C	6	2	6C	6 +	2 +	7C	7	3				M + 1 → M	•	↕	↕	↕	•
JMP					0E	3	2	6E	3 +	2 +	7E	4	3				EA[3] → PC	•	•	•	•	•
JSR					9D	7	2	AD	7 +	2 +	BD	8	3				Jump to Subroutine	•	•	•	•	•
LD	LDA	86	2	2	96	4	2	A6	4 +	2 +	B6	5	3				M → A	•	↕	↕	0	•
	LDB	C6	2	2	D6	4	2	E6	4 +	2 +	F6	5	3				M → B	•	↕	↕	0	•
	LDD	CC	3	3	DC	5	2	EC	5 +	2 +	FC	6	3				M:M + 1 → D	•	↕	↕	0	•
	LDS	10	4	4	10	6	3	10	6 +	3 +	10	7	4				M:M + 1 → S	•	↕	↕	0	•
		CE			DE			EE			FE											
	LDU	CE	3	3	DE	5	2	EE	5 +	2 +	FE	6	3				M:M + 1 → U	•	↕	↕	0	•
	LDX	8E	3	3	9E	5	2	AE	5 +	2 +	BE	6	3				M:M + 1 → X	•	↕	↕	0	•
	LDY	10	4	4	10	6	3	10	6 +	3 +	10	7	4				M:M + 1 → Y	•	↕	↕	0	•
		8E			9E			AE			BE											
LEA	LEAS							32	4 +	2 +							EA[3] → S	•	•	•	•	•
	LEAU							33	4 +	2 +							EA[3] → U	•	•	•	•	•
	LEAX							30	4 +	2 +							EA[3] → X	•	•	↕	•	•
	LEAY							31	4 +	2 +							EA[3] → Y	•	•	↕	•	•

Legend:

OP	Operation Code (Hexadecimal)
~	Number of MPU Cycles
#	Number of Program Bytes
+	Arithmetic Plus
−	Arithmetic Minus
•	Multiply
$\overline{M}$	Complement of M
→	Transfer Into
H	Half-carry (from bit 3)
N	Negative (sign bit)
Z	Zero (Reset)
V	Overflow, 2's complement
C	Carry from ALU
↕	Test and set if true, cleared otherwise
•	Not Affected
CC	Condition Code Register
:	Concatenation
V	Logical or
∧	Logical and
⊻	Logical Exclusive or

Notes:

1. This column gives a base cycle and byte count. To obtain total count, add the values obtained from the INDEXED ADDRESSING MODE table, in Appendix F.
2. R1 and R2 may be any pair of 8 bit or any pair of 16 bit registers.
   The 8 bit registers are: A, B, CC, DP
   The 16 bit registers are: X, Y, U, S, D, PC
3. EA is the effective address.
4. The PSH and PUL instructions require 5 cycles plus 1 cycle for each **byte** pushed or pulled.
5. 5(6) means: 5 cycles if branch not taken, 6 cycles if taken (Branch instructions).
6. SWI sets I and F bits. SWI2 and SWI3 do not affect I and F.
7. Conditions Codes set as a direct result of the instruction.
8. Value of half-carry flag is undefined.
9. Special Case — Carry set if b7 is SET.



IMAGES PRODUCED ON THE ARTRON 2000 STUDIO COMPUTER; COURTESY OF GRAPHIC PRODUCTS, LONDON (EXCLUSIVE DISTRIBUTOR)